PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 17/60

A2

(11) International Publication Number: WO 00/67176

(43) International Publication Date: 9 November 2000 (09.11.00)

(21) International Application Number: PCT/US00/11676

(22) International Filing Date: 1 May 2000 (01.05.00)

(30) Priority Data:
09/304,237 3 May 1999 (03.05.99) US

(71) Applicant: JUST IN TIME SOLUTIONS, INC. [US/US]; Suite 100, 444 De Haro Street, San Francisco, CA 94107 (US).

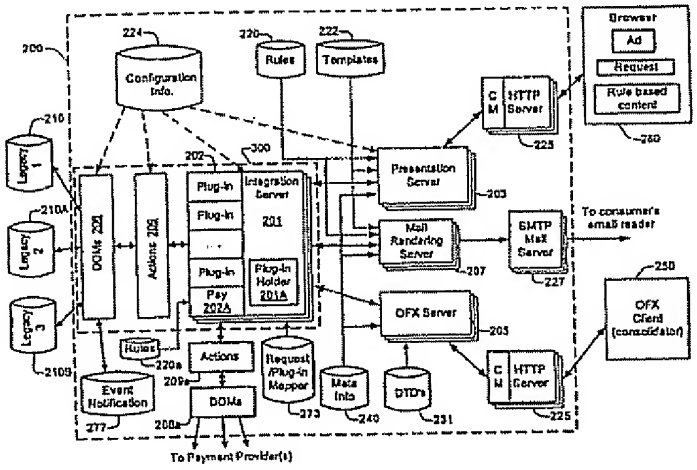
(72) Inventors: DAS, Robin, K.; 66 Bayview Drive, San Carlos, CA 94070 (US). RADOVANCEVICH, Michael, Pavle; 328 Rosilie Street, San Mateo, CA 94403 (US). TWYMAN, Nicholas, M.; 1531 Golden Gate Avenue, San Francisco, CA 94116 (US). BROWN, Mathew; 479 Duboce Avenue, San Francisco, CA 94117 (US). LANZA, Michael; 810 Arkansas Street, San Francisco, CA 94107 (US). VALENTE, Brian; Apartment 1, 1739 Lake Street, San Francisco, CA 94121 (US). DISCHLER, Gerald; 7026 Saroni Drive, Oakland, CA 94611 (US).

(74) Agent: WOLF, Dean, E.; Beyer Weaver & Thomas, LLP, P.O. Box 130, Mountain View, CA 94042-0130 (US).

(81) Designated States: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: TECHNIQUE FOR FACILITATING CUSTOMER TRANSACTIONS OVER A COMPUTER NETWORK USING CUSTOMIZED INFORMATION FROM A BACKEND COMPUTING SYSTEM



(57) Abstract

A technique is provided for dynamically rendering graphical user interface pages over a computer network using customized information from a backend computing system. Using the technique of the present invention, a user is able to send a request for accessing specific information from the backend computing system by making a call on a remote server. In a specific embodiment, the present invention is configured to allow consumers to conduct billing transactions over the Internet. The present invention uses a thin server interface model for interfacing a consumer or user with the customized backend computing systems of one or more specific service providers. The data structure which is used by the user to implement the call request in the remote server may be modified without requiring modifications to the software code in either the client machine or the remote server engine. Further, new applications and/or call requests may be implemented within the system without requiring modification and recompilation of the user and server machine codes.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

TECHNIQUE FOR FACILITATING CUSTOMER TRANSACTIONS OVER A COMPUTER NETWORK USING CUSTOMIZED INFORMATION FROM A BACKEND COMPUTING SYSTEM

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to data transmission over a computer network, and more specifically to a technique for dynamically generating graphical user interface pages over a computer network using customized data from a backend computing system.

Background

Recent advances in computer network technology have paved the way for many service providers to offer their services on-line. This is particularly true with respect to services offered via the Internet. For example, banking services, brokerage services, catalog shopping services, airline reservation services, and a host of other "e-commerce" services are now being offered on-line to consumers via the Internet. Typically, information provided by a particular service provider to an on-line consumer is stored in a backend computing system (often referred to as a Legacy system) operated by that service provider. For example, the service provider may be a bank, and the consumer may be a bank consumer who wishes to access his bank account information via the Internet. The bank account information will typically be stored in a customized backend computing system operated by the bank.

Conventionally, in order for a user to access the information contained within the Legacy database of the service provider, a customized interface must be created (either by the service provider, or by a third party) to interface the user's computer to the Legacy database. This is shown, for example, in FIGURE 1 of the drawings.

FIGURE 1 shows a schematic block diagram of a conventional technique for interfacing a user 102 with a Legacy database or back-end computing system database 150. As shown in FIGURE 1, a customized interface and rendering system 104 is provided for interfacing user 102 with the Legacy database 150.

In applications where the user 102 accesses information from the Legacy database 150 over the Internet, the user's system 102 will typically include a browser (such as, for example, Microsoft Internet Explorer or Netscape Navigator) which communicates with the customized interface and rendering system 104 using HTTP protocol. System 104 includes an interface which has been customized specifically to interface requests received in HTTP protocol (from the user) into the appropriate protocol required for accessing the Legacy database 150. Conventionally, most customized interfaces are implemented using a thick interface model. The thick interface model is typically implemented by providing a customized data structure to the user to make a specific request for accessing information from a Legacy database. The user fills the customized data structure, and implements a call on a remote server, which in the example of FIGURE 1, is system 104. The remote server has been customized specifically to read the data structure provided by the user, interpret the data structure information (including the type of call requested), and perform specific tasks related to the user's call request such as, for example, accessing data or other information from the Legacy database 150.

After the desired information has been retrieved from the Legacy database, server 104 presents this retrieved information to the user 102. In order to present this information, system 104 typically includes a customized rendering system which utilizes a pre-compiled, customized web page or HTML template specifically configured to present the information retrieved from the Legacy system relating to the user's call request.

One disadvantage of the conventional thick interface is that any changes which are desired to be made to the aforementioned data structure (such as, for example, changing the input and/or output parameters associated with a particular call request) typically require modifying and recompiling code in both the client machine 102 and server engine 104. Additionally, in order to present this new information related to the modified data structure, the customized web page(s) or template(s) must also be modified and recompiled. Another problem with the thick server interface is that any new applications which are desired to be implemented also require modifications to the server engine and/or client machine, which may include modifying and recompiling server and/or client machine codes.

Although customized interfaces and rendering systems have been implemented in the past to interface a user with a backend computing system,

continuing efforts are being made to provide improved systems for accessing and presenting data derived from backend computing systems.

SUMMARY OF THE INVENTION

According to specific embodiments of the invention, a technique is provided for facilitating customer transactions over a computer network using customized information from at least one backend computing system. For example, the technique of the present invention may be used to enable such features as: dynamically generating graphical user interface pages over a computer network using customized information from a backend computing system; facilitating communications between a legacy computing system and a graphical user interface system to facilitate the inclusion of customized information from the legacy computing system in at least one graphical user interface page rendered by the rendering system; facilitating automated customer self-care in an on-line billing environment implemented over a computer network; etc. Using the technique of the present invention, a consumer or user is able to access desired information from a customized backend computing system. Further, according to at least one embodiment, the rendering system of the present invention may be fully customizable so that it can be adapted to suit any desired application such as, for example, bill presentment, electronic commerce transactions, account management, etc.

According to a specific embodiment of the present invention, a rendering system is provided for dynamically generating graphical user interface pages over a computer network using customized data from a backend computing system. The rendering system comprises a plurality of page templates, wherein at least some of the page templates include static information and include custom tags which represent customized data associated with data stored in the backend computing system. The rendering system also includes an integration engine configured or designed to access specific information from the backend computing system, and further includes a presentation engine in communication with the integration engine. The presentation engine is configured or designed to render graphical user interface pages using the page templates such that when a first page corresponding to a first one of the page templates is accessed by a user, the presentation engine utilizes the first page template

associated with the first page to render the first page, wherein customized data to be included in the first page is obtained by accessing the integration engine.

An additional aspect of this embodiment provides a bill presentation system comprising a system for dynamically generating the graphical user interface pages described above. The bill presentation system is configured or designed to present on-line bills to a consumer. The bill presentation system further comprises a payment module in communication with the integration engine. The payment module is configured or designed to facilitate payment of bills by the consumer.

The payment module may also be configured or designed to provide the user with dynamic payment tracking information relating to a status of a payment which has been authorized by the user. Additionally, the payment module may also be configured or designed to provides a conditional auto-pay feature, whereby payment of a bill is automatically paid if an amount of the bill is within a predetermined range of values.

An alternate embodiment of the present invention provides a technique for dynamically generating graphical user interface information over a computer network using customized information from a backend computing system. The technique may be implemented as a method of targeting advertising information to a user over a computer network. An on-line itemized bill having at least one billed item is rendered for presentation to a remote user or consumer. Further, a set of rules is consulted to select a particular advertisement to display to the user. The set of rules is based at least in part upon properties or other information related to the on-line bill. The selected advertisement is then rendered for display to the user in combination with the on-line itemized bill. An additional aspect of this embodiment provides that the billing information is obtained by accessing a backend computing system, and that at least a portion of the accessed billing information is used to select the particular advertisement to display to the user.

An additional embodiment of the present invention provides a technique for dynamically generating graphical user interface pages over a computer network using customized information from a backend computing system. A request is received at a first server for selected information to be presented to a remote user. The first server then makes a call request to an integration server to access the selected information. The call request is issued by the first server utilizing a thin interface protocol. The integration server is then used to access the selected information from a backend

computing system, and retrieves output information relating to the call request. The output information is passed to the requesting server, and then rendered for presentation to the remote user. Where the invention is implemented as a computer program product, the computer program product will include a computer usable medium having computer readable code embodied therein for implementing the above-described process.

An additional aspect of the present invention provides a mail rendering system for dynamically generating at least one electronic mail message to a particular customer upon the occurrence of at least one specific event. The mail rendering system may be configured to dynamically generate content of an electronic mail message based upon billing information related to a specific customer.

Another embodiment of the present invention provides a technique for facilitating communications between a legacy computing system and a graphical user interface system to facilitate the inclusion of customized information from the legacy computing system in at least one graphical user interface page rendered by the rendering system. A request is received at a first server for accessing desired information from the legacy computing system. In response to the request, the server may call at least one plug-in object for servicing the request. The plug-in object may call a first action object for handling at least a first portion of the first request. The action object may then call other action objects for handling issues related to that first portion, or may call a domain object for accessing at least a first portion of the desired information from the legacy computing system. An additional aspect of these embodiments provides that a single input parameter is received along with the request. The single input parameter includes tagged, self-described data, which may be used by the server to identify the name of the particular call request to be implemented, and to identify any input parameters associated with the particular call request.

A further embodiment of the present invention provides a technique for dynamically generating graphical user interface information over a computer network using customized information from a backend computing system. A request is received at a first server for selected information to be presented to a remote user. A page template is then selected for presentation to the remote user in response to the request. The page template includes at least one custom tag representing customized data associated with or derived from the backend computing system. A meta file is

then consulted for determining the proper input parameter requirements for making an appropriate call for accessing the customized data. The identified input parameters are then retrieved from a local data pool residing on the server, and sent along with the appropriate call request to access the desired customized data. Where the invention is implemented as a computer program product, the computer program product will include a computer usable medium having computer readable code embodied therein for implementing the above-described process.

An alternate embodiment of the present invention provides a technique for facilitating automated customer self-care in an on-line billing environment implemented over a computer network. An on-line itemized bill is rendered for presentation to a consumer. The on-line bill includes at least one billed item which has an associated billed item link. When the bill item link is selected by the consumer, additional information about the billed item is accessed to help identify or clarify the billed item. This additional information is then provided to the user or consumer. Further, the additional information may be retrieved by accessing the information from a backend or Legacy computing system over the computer network. Where the invention is implemented as a computer program product, the computer program product will include a computer usable medium having computer readable code embodied therein for implementing the above-described technique.

In a specific embodiment of the present invention, the above-described technique of facilitating automated customer self-care renders an on-line itemized phone bill to the consumer. One specific type of customer self care permits a consumer to perform a reverse call look-up for a particular phone number selected by the consumer. In an alternate specific embodiment, an on-line credit card bill which includes at least one transaction record is presented to the consumer. Using the technique of the present invention, the consumer is able to perform a lookup which provides additional information about a specific transaction record selected by the consumer.

Additional features and advantages of the present invention will become apparent from the following description of its preferred embodiments, which descriptions should be taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 shows a schematic block diagram of a conventional technique for interfacing a user or consumer with a Legacy database or backend computing system.

FIGURE 2 shows a schematic block diagram of a specific embodiment of the rendering system 200 of the present invention.

FIGURE 2A shows an alternate embodiment of the rendering system 200 of the present invention in which certain elements or features of the present invention have been omitted.

FIGURE 3 shows a schematic block diagram of the integration server engine 300 of Fig. 2.

FIGURE 4 shows a schematic block diagram of a specific embodiment of the presentation server 203 of Fig. 2, showing how the presentation server interacts with the various other elements of rendering system 200.

FIGURE 5 shows an example of a rendered "log-on" web page.

FIGURE 6 shows an example of a rendered "account summary" page, which has been rendered in accordance with the technique of the present invention.

FIGURE 7 shows an example of a "long distance calls" page which has been rendered in accordance with the technique of the present invention.

FIGURE 8 shows a flow diagram of an example of how a new session procedure 800 is implemented in accordance with a specific embodiment of the present invention.

FIGURE 9 shows a flow diagram for rendering an "account summary page" in accordance with a specific embodiment of the present invention.

FIGURE 10 shows a flow diagram of a render page procedure 1000 such as that identified by block 912 of FIGURE 9.

FIGURES 11A and 11B provide an illustration of how custom tags in the page templates may be used to render customized data specific to a particular user or consumer.

FIGURE 12 shows a flow diagram of a specific embodiment of an integration server process request procedure 1200 which is called, for example, in block 1020 of FIGURE 10.

FIGURE 13 shows a schematic block diagram of a communication path of a specific embodiment of the rendering system of the present invention

FIGURE 14 shows an example of a domain object model "DOM" in accordance with a specific embodiment of the present invention.

FIGURES 15A AND 15B describe the various elements and methods of a specific embodiment of the present invention relating to the Authenticate Sign-on Request procedure 1500 of FIGURE 15B

FIGURE 16 shows an example of a sample meta file information associated with a sign-on request.

FIGURE 16A shows an example of a generated accessor 1650 corresponding with the sign-on request meta information of FIGURE 16.

FIGURE 17 shows en example of how the data pool 426 (FIGURE 4) is used for storing and retrieving information to facilitate web-site flow in accordance with the technique of the present invention.

FIGURES 18, 18A, and 18B describe various features and methods related to the e-mail notification feature of the rendering system of the present invention.

FIGURE 19 shows an example of how the various plug-ins, actions, and DOMs interact in accordance with a specific embodiment of the present invention.

FIGURE 20 shows a flow diagram of a specific embodiment for implementing the payment tracking procedure 2000 of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides a technique for dynamically rendering graphical user interface pages over a computer network using customized information from a backend computing system. Using the technique of the present invention, a consumer or user is able to access desired information from a customized backend computing system or Legacy system. As will be described in greater detail below, the technique of the present invention provides a rendering system which uses a thin server interface model for interfacing a consumer or user with the customized backend computing systems of one or more specific service providers. Using the technique of the present invention, a user is able to send a request for accessing specific information from the backend computing system by making a call on a remote server. The remote server accesses the desired information from the backend computing system using a thin server interface. The data structure which is used by the user to implement the call request in the remote server may be modified without

requiring modifications to the software code in either the client machine or the remote server engine. Further, new applications and/or call requests may be implemented within the system without requiring modification and recompilation of the user and server machine codes.

Additionally, the technique of the present invention also provides a rendering system which includes a Presentation Server for dynamically generating customized web pages for displaying retrieved Legacy information related to the user's call request(s). Unlike conventional rendering systems, the Presentation Server of the present invention may be configured to dynamically and automatically implement desired modifications to the thin interface parameters associated with selected call requests without modifying or recompiling code in either the Presentation Server engine, user machine, or the web page templates which are used by the Presentation Server to present desired information to the user.

In a specific embodiment, the present invention provides a technique for consumers to conduct billing transactions over the Internet. The billing transactions may include, for example, providing billing information to a consumer relating to one or more bills from one or more creditors or institutions (e.g. utility, phone, bank, etc.); providing the consumer with the ability to inspect each of the bills and information contained therein; providing the consumer with the ability to perform customer self-care such that the consumer is able to access additional information relating to specific items of the bill selected by the consumer; showing summary information of a bill; showing detailed billing information of a bill; and providing specific targeted advertisements to the consumer using rules which are based at least in part upon the billing information related to the consumer.

Customer self-care may be further defined by providing the ability for consumers to resolve their problems on-line rather than by calling customer support representatives. This includes, for example, providing the ability for a consumer to perform a reverse call look-up for an unrecognized call listed on the consumer's on-line phone bill.

Another feature which may be implemented by the technique of the present invention is the ability for billing institutions and/or other creditors to increase consumer retention, (i.e., reduce consumer churn) by providing self-enabled services to the consumer, whereby consumers may add or delete services or features to or from their respective accounts. For example, one service which may be self-enabled by the

consumer using the technique of the present invention is the ability to pay a utility, phone or other bill automatically from the consumer's bank account. In theory, the more services provisioned to or utilized by the consumer, the more links the consumer has to his or her account. The greater the number of links the consumer has to his account, the greater the dependence on this account, and the less likely that the consumer will cancel the account. Other types of self-provisioned serviced may be in the areas of mortgage, insurance services, brokerage services, telephone services, etc. In theory, the more services linked to a consumer account, the lower the likelihood of consumer churn.

FIGURE 2 shows a schematic block diagram of a specific embodiment of the rendering system 200 of the present invention. The rendering system 200 may be used to interface a user or consumer 280 with one or more backend computing systems 210. Using the rendering system 200 of the present invention, the user 280 may access desired information from any of the plurality of Legacy computing systems (e.g. 210, 210A, 210B), and after the request for the desired information has been processed, view the desired results in a customized format which has been dynamically generated using templates 222 which include custom tags for presenting the desired data to the user.

In a specific embodiment of the present invention, system 200 includes a plurality of HTTP servers 225 which are used to interface and communicate with external machines, such as a client machine 280 (operating a web browser program), or other type of machine such as, for example, OFX machine 205 which communicates with an OFX client 250 (e.g. a service consolidator) via HTTP server 225. Each HTTP server 225 may communicate with one or more different servers within rendering system 200.

The rendering system 200 also includes one or more Presentation Server(s) 203 for implementing call requests from remote user(s) 280 and for dynamically generating web pages for presenting desired information to remote user 280. System 200 may also include one or more Mail Rendering Servers 207 which may be used for automatically generating e-mail messages to selected consumers via SMTP mail server 227. Additionally, system 200 may include one or more OFX servers 205 for handling call requests implemented by OFX client 250. OFX relates to distribution of information over the Internet or other networks via a consolidated channel. The OFX

client may be, for example, a bill consolidator which provides consolidated billing information to user 280.

When a consumer 280 wishes to access specific information from the Legacy system 210, the consumer sends a request to Presentation Server 203. Upon receiving the request, the Presentation Server 203 responds by rendering an appropriate page template, which may be obtained from either a template directory 222 or a cache within the Presentation Server. If the selected page template contains only static segments, the page may be fully rendered by the Presentation Server and presented to the requesting consumer via HTTP server 225.

Typically, however, a page template will include one or more custom or dynamic tags relating to data which is derived from the Legacy system. When the Presentation Server encounters a custom tag while rendering the selected page template, the Presentation Server first checks to see if the desired data (relating to the custom tag) is stored within a local or internal cache. If the desired data exists within the local cache and is valid, the Presentation Server continues to render the page using the data from the local cache. If, however, the desired data does not exist within the local cache, or exists but is no longer valid, the Presentation Server consults the Meta file 240 to determine an appropriate call request and input parameter(s) for accessing the desired data from the Legacy system. Once obtained, the Presentation Server passes the call request and input parameter(s) to the Integration Server 201.

The Meta Info directory or file 240 provides a global description of what data is available from the Integration Server 201. More specifically, the Meta file completely describes the interface between the Presentation Server and the Legacy system. The Meta file information defines the names of the requests called by the Presentation Server to access information in the Legacy system, and defines the required input parameters, output parameters, and possible error returns related to accessing information within the Legacy system. The Meta file 240 is used by the various servers in system 200 for rendering or distributing data. In a specific embodiment, the Meta file 240 is implemented as a file using an XML-based syntax.

Integration Server 201 is responsible accessing data from the backend computing system(s) and for normalizing data so that the data can be shared between various elements within system 200. In a specific embodiment of the present invention, an XML protocol is used for communicating with Integration Server 201.

Upon receiving a call request and associated input parameters, the Integration Server 201 determines an appropriate Plug-in 202 for handling the call request by consulting Request/Plug-in Mapper block 273.

A request which is made by any of the servers (e.g. Presentation Server 203, OFX Server 205, or Mail Rendering Server 207) to the Integration Server 201 will be mapped to an appropriate Plug-in 202. The Integration Server uses a Request/Plug-in mapping data structure 273 for determining the appropriate Plug-in for handling each particular call request.

Request/Plug-in Mapper block 273 includes a plurality of files which include Meta information similar to the information contained within Meta file 240. Each file within the mapper block 273 is associated with a respective request handler Plug-in 202. The Meta information associated with a particular request handler Plug-in in block 273 includes a complete description of the request and/or other functions which it's associated Plug-in is able to support. The Meta information within each file describes not only the name of the associated request, but also the associated input parameters, output parameters, and/or error messages (if any) associated with each request. The Meta information included within each file of Mapper block 273 may be thought of as a subset of the information contained within MetaFile 240. An aggregate of all the information within each file of Mapper block 273 preferably contains the same information as Meta file 240.

On the backend of the Integration Server are a plurality of Plug-ins 202. Each Plug-in is configured to perform a specific task or set of tasks. Once the appropriate Plug-in for the request has been identified by the Integration Server, the request and input parameters are passed to the identified Plug-in.

The identified Plug-in then calls one or more Actions 209 to perform specific tasks. Conceptually, each of the Actions within block 209 may be thought of as a codification of reusable business logic. Each Action may be implemented as a Plug-in or a group of nested Plug-ins for performing specific tasks under the direction of the top level Plug-in 202. Each of the Actions, in turn, may call other Actions, or may call one or more Domain Object Models (DOMs) 208 for accessing desired data from one or more of the Legacy systems. The DOM provides the interface to the Legacy system, and performs the actual storage and retrieval of information to and from the Legacy database 210. Like Actions, DOMs may also be thought of as a Plug-in or set

of nested Plug-ins for performing specific tasks under the direction of an appropriate Action.

Once the information has been retrieved by a particular DOM 208, the retrieved information is passed back through the appropriate Action(s) 209, and Plug-in 202 to the Integration Server 201. The Integration Server then passes the information back to the requesting server, in this case, Presentation Server 203. The Presentation Server uses this information in conjunction with any appropriate rules and/or templates to dynamically render a customized web page for display to the user 280 via HTTP server 225.

One specific Plug-In identified in FIGURE 2 is a Payment Plug-In 202A, which provides consumers with the ability to implement basic payment transactions regarding on-line bills. The Payment Plug-In has associated with it a corresponding set of business rules 220a, Actions 209a, and DOMs 208a, which have been customized to implement the various payment features available to the consumer. The payment module and its associated rules, Actions, and DOMs are discussed in greater detail in a latter section of this application.

FIGURE 2 also includes a mail rendering server or notification engine 207 which may be used for automatically generating customized e-mail messages to consumers related to the occurrence of specific events, such as, for example, the availability of a new bill at a particular Legacy system. When a specified event occurs, the event is registered in event notification block 277. The mail rendering server 207 periodically checks the notification block to determine whether any new events have occurred. The mail rendering server utilizes the information within rules block 220 and template block 222 to render a customized e-mail message to a specific consumer or consumers relating to the new event. This e-mail message is passed to the specific consumer or consumers via SMTP mail server 227.

System 200 also includes an OFX server 205. The OFX server 205 provides an interface for adding, modifying, and deleting consolidators (OFX clients). Additionally, the OFX server provides an interface for adding, modifying, and deleting biller entities (e.g., Legacy systems). As commonly known to those skilled in the art, biller entities typically provide a bill which is published to a consumer. The biller entity is typically associated with one or more backend computing systems 210. The OFX client or consolidator 250 provides a forum for presenting a bill from the biller entity to the consumer. An example of an OFX client could be a banking

institution which provides consolidated on-line billing information to its customers. The OFX server 205 uses information from DTD (Document Type Description) block 231 for rendering information which is presented to the OFX client via HTTP server 225.

Also included in rendering system 200 is a configuration information block 224 comprising configuration information which is used to configure the various elements within the rendering system 200 upon initialization or start-up. Once the initial configuration of the system has been completed, and the system is up and running, the system 200 is able to perform a variety of tasks, as described in greater detail below.

It will be appreciated by those having skill in the relevant art that the rendering system 200 of FIGURE 2 shows one specific embodiment of the present invention. It is by no means the only embodiment for implementing the technique of the present invention. Various components or elements within system 200 of FIGURE 2 may be added or omitted, as desired, for implementing selected features of the present invention. For example, FIGURE 2A shows an alternate embodiment of the present invention in which certain features of the present invention have been omitted. Where specific features of the present invention are not desired, any or all of their associated elements may be omitted while still achieving the above-described advantages of the present invention. For example, if it is not desired to provide automated e-mail rendering services to consumers, Mail Rendering Server 207 and SMTP mail server 227 may be omitted.

Returning to Fig. 2, each server within rendering system 200 (e.g. Presentation Server, Integration Server, Mail Server, OFX Server, etc.) shares a number of similar characteristics. First, each server is run-time configurable, meaning that it is able to be configured while it is running. Second, the status and performance of each server may be monitored. Third, each server is componentized, providing it with the ability for modular upgrades, integration, management, etc. Fourth, each server supports multi-machine configurations with different types of load balancing. In addition, each server is multi-threaded to make use of SMP (Symmetric Multi-Processor) architectures.

The rendering system 200 may include hardware and/or software for dynamically generating graphical user interface pages over a computer network using customized data from a back end computing system. The backend computing system

(herein referred to as a "Legacy" computing system) may include one or more databases identified in FIGURE 2 by reference numbers 210, 210A, and 210B. Examples of Legacy systems include utility and telephone company databases (e.g., AT&T, Pacific Bell, PG&E), institutional databases (e.g., banks, airlines, etc.), or other proprietary/standardized back end computer systems comprising data which is to be accessed by a remote user over a computer network.

For purposes of simplification, the technique of the present invention will be described in terms of rendering HTML based web pages over the Internet. However, it will be appreciated by those skilled in the art that the technique of the present invention may be used to render any type of information to a user via graphical user interface over a computer network.

The user (otherwise known as the consumer) is represented by block 280 of FIGURE 2. Typically, the user will be a person operating a computer system which is linked to a computer network such as, for example, the Internet. The consumer machine 280 includes a browser for rendering and displaying graphical user interface pages (e.g., web pages) on a computer display. Typically, a graphical user interface page (such as a web pages) will include an advertisement and information relating to a request by the user. The information may include customized information which is tailored specifically to a particular use. Additionally, the information may include links to other sources of information which may provide a more detailed explanation of the linked items.

In a specific embodiment of the present invention, client machine 280 is a computer system which includes a web browser for interfacing with the Internet. Client machine 280 communicates to an HTTP server 225 to access web pages over the Internet. The HTTP server 225 communicates with Presentation Server 203. A primary function of Presentation Server 203 is to render graphical user interface pages to server 225 using templates stored within template directory 222.

In a specific embodiment of the present invention, template block 222 includes a plurality of HTML templates. Each template may include at least one static portion and/or at least one customized portion. The information within the static portion of the template does not change or vary, and therefore will typically produce the same static object when rendered. An example of a static portion is the background image name which, when rendered, will appear as an object on the rendered page. In contrast, the customized or dynamic portion of the template, when rendered, will

produce customized objects or other information on the web page which typically will vary depending upon selected properties. For example, the customized or dynamic data could be detailed billing data specific to a particular user.

When the Presentation Server encounters a custom tag, it renders the dynamic data associated with the custom tag by first checking to see if the data is stored within a local cache. If the desired data is not found in the local cache, the Presentation Server sends a call request to the Integration Server 201 to retrieve the desired data from the Legacy database. A primary function of the Integration Server 201 is to interface with the Legacy database(s), access desired information as specified by a requesting server (e g. Presentation Server, Mail Rendering Server, OFX Server, etc.), and when necessary, return the result of any accessed information to the requesting server. Further descriptions of the functions of the Integration Server are presented in latter sections of this application.

The rendering system of FIGURE 2 also includes a rules directory 220 and a template directory 222. Rules directory 220 includes at least one set of rules or business logic which is used by the various servers (such as Presentation Server 203) for determining what information to render for display to a particular user. Template directory 222 includes at least one set of templates (such as, for example, HTML templates) which are used by the various servers for rendering specific pages related to specific information which is to provided to a particular user. The Template directory 222 may implemented as a file or folder comprising a plurality of individual templates, or a database which comprises a plurality of individual template files. Rules block 220 may also be implemented in a similar manner..

Rules block 220 may include one or more set of rules used by the Presentation Server to determine specific information to render for a particular page. A rule represents a codification of executable business logic, which, when executed, returns a textual result. This business logic can be added and modified independently of the code that runs the Presentation Server. Rules may be implemented via custom tags embedded within the HTML templates, and may be used for a variety of purposes including customizing the "look and feel" of the web-page based upon user profile characteristics; presenting advertisements to users based upon consumer profiles, billing, or any other data available from Integration Server; changing the flow of the web-site based upon particular data; customizing e-mail messages; modifying payment options based on a user's ability to pay; and other uses described herein.

A simple example of a rule may be to render an image of a California flag on a consumer's Account Summary page, but only if the consumer lives in California. Another application of the rules feature relates to rendering customized advertisements based at least in part upon specific consumer billing information. For example, let us suppose that we are interested in attempting to retain highly valued west coast consumers with an overseas calling plan. A rule may be constructed using the following guidelines. First, a highly valued consumer is one who lives in California, Nevada, Oregon, or Washington, and has had an average six-month bill amount each month of greater than $100. Additionally, a highly valued consumer is one that has not been late in paying his or her bill more than twice in the last two years.

If any of the highly valued west coast consumers call the Pacific Rim and the total average cost of those calls is greater than 50% of their total bill, than a specific advertisement may be selected to be rendered on the Bill Summary page of these consumers' bill. For example, the selected advertisement may be read "We will reduce your Pacific Rim calls by 50% if you pre-pay for six-months of phone service." Thus, using consumer specific billing information, the rules may be used in conjunction with the Presentation Server to present customized advertisements to the end-user.

Third party software products may be used for generating or constructing rules within rules directory 222, such as, for example, Neuron Data Elements Advisor, from Neuron Data, Inc., of Mountain View, California. In a specific embodiment of the present invention, the rules may be deployed via an HTML graphical user interface. This graphical user interface (GUI) is not accessible to consumers or other persons who do not have administrator access to the rendering system of the present invention. The rules physically reside in a shared file-system. Sets of rules are placed into two buckets, namely, deployed and in-progress. Deployed rules are those which have been pre-compiled and are available to end-users on the site. In-progress rules are those currently be worked on. For example, these rules may have already been compiled and are currently being tested. They are not available to end-users on the site until they have been deployed via the GUI.

Rules may be deployed to multiple Presentation Servers simultaneously, with each server's performance characteristics being taken into consideration. Deployed

rules are loaded at server-start-up time, and are dynamically loaded and unloaded as they are deployed or undeployed.

Because a rule is stateful, it is preferable that concurrent execution of a rule (by more than one single end user) be avoided. Accordingly, in a specific embodiment of the present invention, a rules pool is provided to overcome this limitation. The rules pool may include many copies of each pre-compiled rule. When a user executes a particular rule, it finds an available copy of that rule in the pool. When the rule is finished executing, it is reset and returned to the pool. If there are no more rules available in the pool, the user is blocked until one returns or becomes available. The number of compiled copies of rules in the pool is dynamically optimized based on monitoring real-time usage patterns over time. For example, rules on higher hit-rate pages may require more copies in the rules pool than those deeper within the web-site. This optimization is automatically maintained by the system based on heuristics.

Integration Server Engine

FIGURE 3 shows an example of the integration server engine 300 of FIGURE 2 of the drawings. The complete integration server engine 300 may be thought of as including the Integration Server 201 (e.g. "thin" interface server), Plug-ins 202, Actions 209, and DOMs 208. When the presentation server 203 desires to access specific information from the Legacy system 210, it sends an appropriate call request along with any input parameters to the integration engine 300. The request is received at the Integration Server (e.g. "thin" server) 201. One of the unique features of the present invention is that the Integration Server is configured to utilize a "thin" server interface, as opposed to conventional rendering systems which utilize a "thick" server interface. The distinctions between "thick" and "thin" server interfaces are described in greater detail in a latter section of this application.

Once the Integration Server 201 receives a call request from a server, the Integration Server interprets the call request, and identifies an appropriate Plug-in 202 for handling the request. The Integration Server then passes the input parameters of the call request to the identified Plug-in 202 to handle the request.

Each Plug-in may be configured to call one or more Actions 209 for handling specific tasks related to the call request. Each Action, in turn, may call one or more other Actions for handling other specific tasks related to the call request. Alternatively, an Action may call one or more DOMs 208 in order to access specified

data within the Legacy system 210. The relationship between Plug-ins, Actions, and DOMs are described in greater detail below.

The concept of a Plug-in is generally known to those skilled in the art. It represents an object which may be incorporated by or coupled to another object. Each Plug-in object is typically configured to implement a specific set of tasks or functions. Plug-ins are used for encapsulated logic and are treated as units. Each Plug-in may be independently configurable. Additionally, each Plug-in is loaded at start-up time and/or dynamically loaded at run-time. Examples of the plurality of Plug-ins 202 identified in FIGURES 2 and 3 may include a Payment Plug-in, an E-mail Plug-in, and one or more request-related Plug-ins for interfacing with the Legacy system. As shown in FIGURE 3, Integration Server 201 includes a Plug-in holder block portion 201A which maintains at least one list of different types of Plug-ins 202 that are coupled to the Integration Server.

At least a portion of the plurality of Plug-ins 202 communicate with one or more Actions, represented by Actions block 209. Each Action represents a business logic component or object for accomplishing a specific task related to a specific Plug-in procedure. For example, one Plug-in 202 to the Integration Server may be called "Add Funding Account". The Plug-in may call, in turn, a plurality of individual Actions 209 related to the process of adding a funding account. These Actions may include, for example, (1) determine funding account type; (2) validate funding account; (3) encrypt funding account; and (4) save funding account. Each Action is composable and designed to be reusable whenever needed. A more detailed description of the relationship between Plug-ins and Actions is provided in the discussion of Fig. 19.

Different types of Actions may include single, composite, and conditional Actions. A single Action is one that does not include or call any other Actions. A composite Action is a set of nested Actions. A conditional Action is one which executes a set of Actions after executing a first set of Actions. The conditional Action may then execute other appropriate Actions based on the result from the initial or first set of Actions.

Each Action communicates with one or more Domain Object Models (DOMs), represented in FIGURE 2 by block 208. A DOM provides an interface between the Integration Server and the Legacy system database. In a specific embodiment of the present invention, an Action may call a DOM in order to access

specific data within the Legacy system. Data passed from the Action to the DOM may be implemented in XML format. The DOM manipulates the data into a format required for interfacing with the Legacy system. Input parameters are passed to the Legacy system, and output data is retrieved. The DOM manipulates this output data back into XML format and passes the retrieved data back to the requesting Action. The Action, in turn, passes the retrieved data to the requesting Plug-in 202 (either directly or via other Actions). The requesting Plug-in than passes the retrieved data to the Integration Server 201, which forwards the retrieved data to the Presentation Server 203.

Both Actions and DOMs may be implemented as Plug-ins which are configured to perform specific tasks. One advantage of implementing plug-in Actions and plug-in DOMs within the integration engine 300 is that each plug-in type serves a different purpose and may be configured separately without changing the other. For example, Actions typically implement business logic, which may be changed or modified or added without necessarily changing the physical storage logic. DOMs, on the other hand, typically represent physical storage logic, which may be changed or modified without necessarily changing the business logic. Thus, by utilizing both Actions and DOMs, the rendering system of the present invention is provided with greater flexibility.

An example of a specific embodiment of a Domain Object Model (DOM) is shown in FIGURE 14 of the drawings. As shown in FIGURE 14, DOM 1400 includes a Meta Data Descriptor field 1402, a Request-type field 1404, and an Instance field 1406 for "caching" data to be input to or retrieved from the Legacy system. The Meta Data Descriptor field 1402 provides a description of Legacy data which DOM 1400 is able to access. Typically, one of the data fields will be assigned as a "key" for retrieving the other data. For example, the consumer's social security number (SSN) may be used as a key to access data relating to the consumer's name, birthdate, account number, etc.

The Request Types field 1404 provides a description of the various actions which DOM 1400 is able to perform for the Legacy data described in field 1402. These actions typically relate to call requests which DOM 1400 is able to make when accessing information within the Legacy system. The instance field 1406 provides an instance of data which is either to be sent to the Legacy system, or which has been

retrieved from the Legacy system in response to a particular call request sent by DOM 1400.

The information contained within the Request field 1404 and Instance field 1406 may vary, depending upon the key used within the MetaDataDescriptor field 1402. In the example of FIGURE 14, the consumer's social security number is used as the key. Based upon this key, the various requests (to the Legacy system) related to this key are shown in Request field 1404. Similarly, the instance data related to this key is shown in field 1406.

FIGURE 19 shows an example of how the various Plug-ins, Actions, and DOMs interact in accordance with a specific embodiment of the present invention. The Presentation Server (not shown) sends an "Add Funding Account" request to the Integration Server. The Integration Server identifies the request, and initiates the most appropriate Plug-in for implementing the request. In the example of Fig. 19, the Integration Server calls the Plug-in "Add Funding Account" 1904. This Plug-in is actually a series of nested Plug-ins for adding a funding account for a specific consumer. The "Add Funding Account" Plug-in 1902 calls the "Determine Funding Account Type" Action 1904. Thereafter, Plug-in 1902 calls "Validate Funding Account" Action 1906. The "Validate Funding Account" Action includes two sub-Actions: "Validate Format" Action 1908, and "On-line Check" Action 1910. The "On-line Check" Action 1910 calls "Credit Card Verification" DOM 1916. This DOM (1916) performs an on-line check to verify credit card information associated with the user/consumer. Once verified, DOM 1916 passes the results back to Action 1910. Thereafter, the "Add Funding Account" Plug-in 1902 calls "Encrypt funding account" Action 1912. After the funding account has been encrypted, Action 1914 is called to save the funding account. This Action makes a call to Funding Account DOM 1918 to store the funding account data within the Legacy system database 210.

PRESENTATION SERVER

FIGURE 4 shows a schematic block diagram of a specific embodiment of the Presentation Server 203, showing how the Presentation Server interacts with the various other elements of rendering system 200.

As shown in FIGURE 4, the Presentation Server 203 includes a global scope portion 450 and one or more session scope portions 415. In the specific embodiment

shown in FIGURE 4, only one instance of the global scope portion is provided for each respective Presentation Server 203, while multiple instances of session scope portion 415 are provided. Each instance of the session scope portion corresponds to a session of a respective, concurrently logged-on user.

The various functions of the global scope portion 450 are represented by the various blocks within that portion. In a specific embodiment, each of the blocks shown within the Presentation Server are implemented via software. Alternatively, a combination of software and hardware may be used.

Session management block 452 is responsible for managing data for each individual session 415. This block maintains a state or session for each individual consumer which is concurrently logged on to the Presentation Server.

Nucleus (name space) block 454 is responsible for maintaining a data structure of services by name provided by the Presentation Server 203. Each portion of code associated with the Presentation Server may be accessed by looking up the service name within nucleus block 454. New services which are to be added to the Presentation Server may also be created within nucleus block 454 for future reference.

Rules engine block 456 is responsible for storing and managing compiled copies of the rules (from block 220) and for maintaining the rules pool (discussed previously with respect to rules block 220). Page cache 457 is responsible for caching static portions of rendered pages associated with template block 222 (FIGURE 2).

The page compilation block 458 is responsible for storing and executing compiled portions of template pages. In this regard, it is responsible for storing pre-compiled code of custom tags associated with the templates within block 222. Page compilation block 458 is also responsible for storing and retrieving servlets. As commonly known to one having ordinary skill in the art, servlets are compiled portions of customized Java code which are embedded within the HTML templates. At rendering time, they are executed, and the results of the executed code replaces the actual Java code tags in the HTML.

As described previously, one or more of the HTML templates within template block 222 may be processed by the Presentation Server 203. The custom tags which identify the customized information within each template is replaced during processing by calls to predefined code, such as, for example, Java code. The Java code is compiled the first time that the page is rendered. From that point on, the compiled class is simply executed. The compiled code is stored within the page

compilation block 458 of the Presentation Server so that, in subsequent calls to this template, the compiled code is simply executed rather than having to be recompiled. During page rendering, the output from the Java code replaces the customized tags in the template. The final result is sent back to the browser via the HTTP server 225.

As will be described in greater detail below, the compiled code of custom tags may be used to call customized droplets which are used for rendering data, and presenting the rendered data according to a specific format.

When a user logs on to the Presentation Server via HTTP server 225, a new session 415 is created for that user by allocating a block of memory to that user on the server. This memory can then be accessed across the user's entire "session" with the server, until the session is terminated. A session may be terminated, for example, either explicitly, or by the session timing out.

Each session includes a data pool 426 which is responsible for storing selected information used to facilitate transitions between web pages in order to avoid the tedious task of writing customized code for handling parameters which are passed between each and every combination of page templates. The data pool 426 is a keyed repository for data that is used for passing the output of one request into the input of another. The data pool facilitates flow from page to page by storing inter-page data which is needed to survive from a first page to a subsequent page.

Each session instance 415 also includes an Execution Cache 420 which is responsible for storing selected dynamic data retrieved from the Integration Server. One purpose of storing dynamic data retrieved from the Integration Server in the Execution Cache 420 is to minimize the number of calls made to the Integration Server by the Presentation Server, which, in turn results in faster response times for rendering pages to the user. When the Presentation Server receives results back from the Integration Server relating to a particular call request, it caches all non-volatile data retrieved from the Integration Server 201 in Execution Cache 420. When a call request is made by the consumer to access information from the Legacy system, the request is first passed to the execution cache. If the requested data is stored within the execution cache and is valid, the execution cache is able to present the requested data to the consumer without performing an access step to the Legacy system via the Integration Server, thereby greatly reducing the turn around time for the call request. The retrieved data will then be passed to page rendering block 431 for processing.

Additionally, each session portion 415 of the Presentation Server includes a Page Rendering portion 431, which represents a running instance of droplet classes that have been compiled and stored within system 200. A droplet is at custom tag that is used for performing complex rendering actions such as looping, sorting, or switching. Each droplet provides code for instructing the Presentation Server on how to present data to the HTTP server 225. Moreover, each droplet is reusable, meaning that the same droplet can be applied against different data.

A generic droplet framework may be implemented via 3rd party vendor products such as, for example, ATG Dynamo (manufactured and licensed by Art Technology Group of Boston, Mass.), which provides basic droplets for rendering HTTP web pages using HTML templates. Using the generic droplet framework, the present invention has been configured to include a plurality of unique, customized droplets, at least some of which are described in greater detail below.

Examples of some of the various droplets utilized by the rendering system 200 of the present invention include formatting droplets, rule-execution droplets, presentation droplets, etc. The presentation droplets include a ForEach droplet and a Switch droplet, among others. The Switch droplet allows for different HTML paths based on the value of a data item. The ForEach droplet loops through one or more lists of data and provides options for (1) column display order; (2) ascending and descending column-level sorting (which may be switched by the user by clicking on a column header); (3) default sorting order; and (4) automatic paging. The ForEach droplet may be applied against any tabular data to provide a sorted HTML table. The tabular data, which, for example may be data retrieved from the Integration Server, may be sorted in an order according to a sorting key selected by the consumer. An addition feature of the ForEach droplet reverses the sorting order if the same sorting key is re-selected by the user.

Rule droplets execute specific rules and place the textual result of the rule (e.g., image name, HTML, text, etc.) on the rendered page. Form handling droplets execute requests by using form fields as input parameters to specific requests. In a specific embodiment, the rendering system of the present invention uses a three-level formatting system. Page level formatting will format only a specified instance on a page. Field-level formatting, if not overridden with page-level formatting will format that particular field anywhere on the web-site. Type-level formatting, if not overridden by page or field-level formatting will format all fields of a specified type.

Each session 415 also includes a data accessors block 421, which comprises a plurality of data accessors. Block 421 represents running instances of accessor classes for a particular session. Each data accessor converts or translates a specific user request into related methods or functions which are then passed as calls to the execution cache 420. Accessors are objects typically written in Java code (or other programming language) which are used by the presentation engine to render information using the graphical user interface templates. The accessors are generated by an accessor generator program (not shown). The accessor generator program comprises an executable code which creates accessors by parsing the information contained within the Meta file 240.

FIGURE 16 shows sample Meta file information associated with a Sign-on request. As shown in FIGURE 16, the Meta file information includes the name of the request 1602 (Sign-on); the type of request 1603 (data); input parameters 1604 for the Sign-on request; output parameters or return data 1606 from the Sign-on request; and possible error returns 1608 related to the Sign-on request. Further examples of the use of the Meta file information are described in greater detail in later sections of this application. In a specific embodiment of the present information, the Meta file information is stored in XML format. The accessor generator is able to read the XML format and translate this information into Java code or "Java Beans", commonly known to those skilled in the art.

The accessor generator is designed to read the Meta file information (such as that shown in FIGURE 16), and create individual accessor objects relating the various calls and their associated parameters described within the Meta file. Using FIGURE 16 as an example, the accessor generator will identify line 1602 as a call named "SIGN-ON" Using this information, the accessor generator will create an accessor associated with the SIGN-ON request. This accessor is typically written in Java code, HTML, or other similar-type computer mark-up language. As the accessor generator continues reading the Meta information shown in FIGURE 16, it recognizes the various parameter labels (e.g., in, out, status, etc.) and generates the appropriate Java code for instructing the Presentation Server on how to handle data sent to and received from the Integration Server relating to the SIGN-ON request. Thus, in the example of FIGURE 16, the SIGN-ON accessor will comprise code which includes a SIGN-ON request to be sent to the Integration Server along with input parameters LOG-IN and PASSWORD, and will further include code for receiving output data

from the Integration Server identified as CustomerID. Additionally, the SIGN-ON accessor may include code for handling errors related specifically to the Sign-on request.

FIGURE 16A shows a simplified example of a specific embodiment of a generated Accessor related to the Sign-on call request. This generated Accessor will reside, for example, in the Data Accessors block 421 of the Presentation Server (FIGURE 4). The Sign-on Accessor is generated by running a accessor generator program which uses the information contained within MetaFile 240. In a specific embodiment of the present invention, Accessors are initially generated using the accessor generator program, after the rendering system 200 has been integrated with the Legacy system and all information within MetaFile 240 has been completed. Additionally, any time the MetaFile information within block 240 is modified, it is preferable that new Accessors be generated using this new MetaFile information by running the accessor generator programmer again. This may either be implemented automatically or manually by the system administrator.

The Accessor of 16A was generated using the Meta Information shown in FIGURE 16. Accessor 1650 includes a class name description 1658 ("Sign-On") describing the name of the associated call request. Also included within the Sign-On Accessor 1650 are a plurality of methods or functions specific to the Sign-on call request. Elements 1652 and 1654 represent functions or methods for setting up input parameters (LoginID, Password) which will be passed to the execution cache 420. Element 1656 represents a method or function for retrieving from the execution cache 420 the CustomerID data associated with the LoginID and password input parameters.

The technique of using an accessor generator to create accessors for interfacing the Presentation Server with the Integration Server is an innovative and novel concept which provides the rendering system of the present invention great flexibility with regard to customizing and modifying the rendering of customized data stored in a Legacy or other backend-type computing system. As stated previously in the background of this application, conventional rendering systems which interface a client with a backend computing system are implemented using a "thick" interface, which is commonly known to those skilled in the art.

Using the thick interface, a client fills the customized data structure, and implements a call on a remote server. The remote server has been customized for a specific application or task. When the server receives the remote call from the user, it

unpacks the data from the data structure. The customized server is able to read the customized data structure and perform specific tasks related to the client's call request. The disadvantage to this thick interface is that changes to the data structure requires changes to the interface description language (IDL). This, in turn, requires modification and recompilation of code in the client machine and/or server engine. Additionally, if new applications are desired to be added into the system, this also requires product engine changes which may also necessitate modifying and recompiling client and server codes. These product changes are undesirable because they typically require the client to purchase and install upgrades of code on the client machine(s).

In contrast, the technique of the present invention presents a "thin" server interface, which is shown, for example in Fig. 3 of the drawings. Rather than designing specific code for each call request which may be implemented by the Integration Server, the technique of the present invention uses a Meta file 240 which completely describes all parameters necessary for interfacing or communicating with the Integration Server. An accessor generator is used to read the entire Meta file information and generate a plurality of accessors relating to each specific call request and related parameters identified in the Meta file. The generated accessors are used for implementing the thin interface model. One reason why this model is referred to as a thin interface is that the Presentation Server need only send a single string data type to the Integration Server. Included within this single data string is the name of the request to be executed (e.g., GetSummaryData), and all of the relevant "tagged", self-described data associated with that request.

Under the thick interface model, however, the client sends a specific request to be implemented by the remote server. For example, a client may send the request GetSummaryInfo to the remote server. On the server end, when this request is received, the customized server code will recognize the call request and interpret the data passed to it according to a predetermined format. If the received data is not formatted precisely as the server expects it, an error will result. Further, if it is desired to modify the input parameters associated with a specific call request, the code relating to this request must be modified in both the client and server and recompiled.

In contrast, according to the technique of the present invention, to implement a GetSummaryInfo request, for example, the Presentation Server sends a single string data type to the Integration Server. The Integration Server looks up the request name

field within the data to identify the particular request to be executed. Since the request parameters for the thin interface are each tagged and described, the Integration Server knows how to interpret the request data by referencing the tagged descriptions of the data. This feature provide great flexibility.

If it is desired to modify the parameters associated with the GetSummaryData request, the Integration Server is modified accordingly. More specifically, a specific Plug-in 202 (which is coupled to the Integration Server) for implementing the GetSummaryData request will be modified. As will be described in greater detail below, a plurality of Plug-ins 202 are coupled to the Integration Server for handling dynamic data access requests from the Presentation Server. Each Plug-in includes a portion of Meta information describing the specific request(s) and related parameters handled by that Plug-in. For example, an Authentication Plug-in may handle a Sign-on request from the Presentation Server The Meta information contained within the Authentication Plug-in will include the sample Meta information shown in Fig. 16 relating to the Sign-on request. In a specific embodiment of the present invention, the Meta file 240 is an aggregated list of all the Meta information contained within each Plug-in 202. Thus, when modifications are made to a particular Plug-in, these modifications are reflected in the Plug-in's Meta information and also in the Meta file 240.

Once the modifications to the Meta file 240 have been implemented, the accessor generator is executed for reading the modified Meta information relating to the modified request parameters (e.g. GetSummaryData), and generating new accessors incorporating these modified changes. When the Presentation Server desires to make a GetSummaryData call to the Integration Server, it simply sends an "execute" request to the Integration Server in the same manner it did before, passing a single parameter which includes self-described, tagged data describing the new modified data in accordance with the new parameter descriptions in the Meta file. The Integration Server will then interpret the request data by referencing the tagged descriptions of the data, and will execute a GetSummaryData request. No modification to the Presentation Server or the Integration Server code need be made.

Another powerful and advantageous feature realized by the technique of the present invention is run-time introspection of the Integration Server interface. For example, in a specific embodiment of the present invention, a "GetMetaInfo" request may be sent to the Integration Server for causing the Integration Server to output all

Meta information relating to its interface. Thus, for example, if the GetSummaryData requests in the Integration Server has been modified to include additional parameters, these additional parameters will be described when responding to the GetMetaInfo request. In a specific implementation of the present invention, the GetMetaInfo request may be used to generate a new Meta info file 240. This feature may be used for run-time testing of the system, or used, for example, by locator block 440 (Fig. 4) to locate an appropriate Integration Server for handling a particular call request.

Also shown in FIGURE 4 is a locator block 440. Locator 440 provides location and load balancing management between the Presentation Server and the Integration Server. It may be implemented as a separate process which may run on its own machine, or on another machine within the system 200. In specific embodiments of the present invention, where a plurality of Integration Servers exist, locator block 440 periodically checks the load of each Integration Server, and announces this to the Presentation Server(s). Further, the locator 440 provides each Presentation Server with information relating to Plug-ins which are installed or coupled to each respective Integration Server. It is preferable that the Presentation Server communicate with an Integration Server that has proper Plug-ins for servicing specific request. Since the rendering system 200 of the present invention may include a plurality of Presentation Servers and a plurality of Integration Servers, one or more locator blocks 440 may be provided to direct specific Presentation Servers to appropriate Integration Servers, as needed. Further, to handle administration, each Presentation Server has its own built-in HTTP server (not shown) for administration and testing.

In a specific embodiment of the present invention, the Presentation Server 203 is based on an implementation of ATG Dynamo (manufactured and licensed by Art Technology Group of Boston, Mass.), which is a generic software program for rendering HTTP web pages using HTML templates.

FIGURE 6 shows an example of a graphical user interface page which has been rendered in accordance with the technique of the present invention. The rendered page of FIGURE 6 includes both static objects and customized objects. For example, the title of the page, "Account Summary" 610 is a static object which is displayed to every user accessing this page. Portions 606 of FIGURE 6 represents customized data which will differ for each user.

The web page of FIGURE 6 is rendered by the Presentation Server 203 using a template within template directory 222 of FIGURE 2. The customized portion of the

template which represents objects 606 and 612 (FIGURE 6) includes custom tags representing dynamic data which is to be retrieved from the Legacy database 210.

Examples of custom tags used in accordance with a specific embodiment of the present invention are shown in FIGURES 11A and 11B of the drawings. FIGURE 11A of the drawings shows an example of a rendered web page 1100 as displayed by a user's browser program. The customized information identified at 1104 was rendered by the Presentation Server using the custom tags 1104a and 1104b of Fig. 11B.

FIGURE 11B shows an example of a customized or dynamic portion of an HTML template from template directory 222. Examples of two custom tags 1104a and 1104b are shown in FIGURE 11B. A first custom tag 1104a is a "value of" tag which calls an AccountSummary Accessor instance within that particular session, invoking the GetName method of that Accessor. The specific method of the AccountSummary Accessor which is called by custom tag 1104a checks the execution cache 420 for account summary "name" data. If this data is not found in the execution cache 420 of the Presentation Server (Fig. 4), it then sends a fetch request to the Integration Server to retrieve this data.

Similarly, custom tag 1104b calls the GetAccountNo method of the AccountSummary Accessor. This method first checks the execution cache for data stored under AccountSummary.AccountNo, and, if not present, sends a fetch request to the Integration Server to retrieve this data, for example, from the backend computing system. Once the dynamic data for these custom tags has been retrieved, it is rendered by the Presentation Server and displayed to the user as shown in 1104 of FIGURE 11A.

Every page transition at the consumer node results in a different HTTP request to be processed by the Presentation Server 203. In order to facilitate web-site flow, a data pool 426 (Fig. 4) is provided for each designated session. The data pool is a session-level object that allows the Presentation Server to save a request's output data so it can be used as input data to a request on another page. The presentation system can automatically store and retrieve data to and from the data pool. Additionally, the web-site author can explicitly store and retrieve data to and from the data pool by using custom tags. The use of the data pool eliminates the need for a programmer to write customized code for each page in the web-site for forwarding session-level data from one page to another page.

FIGURE 17 shows an example of how the data pool 426 is used for storing and retrieving information to facilitate web-site flow. Fig. 17 provides a simple example of website flow that, in traditional systems, would typically require custom back-end coding to flow from page-to-page. Using the technique of the present invention, no coding is required. Each page transition represents a completely different HTTP transaction to the HTTP server and consequently, the Presentation Server. In order for the output data from a first request to be stored so that a next request can use this data as input parameters, the output data is stored in the data pool of the Presentation Server.

At 17A, the consumer is presented with a Sign-on page. After the user has entered a log-in ID and password, this information is passed to the Presentation Server 203. The Presentation Server forwards this information to the Integration Server, which uses the input parameters (log-in ID, password) to retrieve specific data from the Legacy system. The Legacy data may include, for example, an AccountID associated with that particular consumer. The AccountID is passed from the Legacy system to the Integration Server to the Presentation Server where it is stored within data pool 426.

The next page in the web-site flow is the Account Summary age, which is identified in FIGURE 17 as page 17B. The Account Summary page looks up the current account summary information for this user by using the user's AccountID as a key. Thus, the input parameter for the Account Summary Page is the AccountID. In rendering page 17B, the Presentation Server retrieves the AccountID information from the data pool and passes this input parameter along with a GetSummaryData request to the Integration Server. Using this input parameter, the Integration Server responds to this request by providing output data including the name, address, state, zip, account number, StatementDate, due date, and amount due information to the Presentation Server 203.

This information is displayed to the consumer as shown, for example, on page 17B. Any of these output parameters which are to be used as input parameters for subsequent web pages may be stored within the data pool. In the example of FIGURE 17, the StatementDate is placed into the data pool since it used as one of the input parameters for the Bill Details page 17C.

In order to determine which particular output data to store in the Data Pool 426, the Execution Cache 420 consults the Meta file info 240. The Meta file

information includes a key or tag associated with each output parameter (of a particular call request) for indicating whether that specific parameter is to be automatically stored in the data pool 426. In the example of Fig. 17B, the Meta file information (not shown) indicates that only the statementDate data is to be stored in the Data Pool.

When the user clicks on the details button on the Account Summary Page (17B), the Presentation Server responds by sending a GetDetailedData request to the Integration Server along with input parameters AccountID and StatementDate. The AccountID and StatementDate parameters are retrieved from the data pool and sent as input parameters to the Integration Server.

The Integration Server responds by providing output data relating to the Bill Details page which, in this example, is an array of dates, numbers, minutes, and amounts, as shown on the bill details page (17C) of FIGURE 17.

PAYMENT MODULE

As described previously, rendering system 200 may be configured to include a payment plug-in module, which enables consumers to initiate payment of on-line bills. The payment module provides a variety of novel features. First, the payment module provides the ability for each consumer to access multiple payment accounts. Examples of different payment accounts for a particular consumer may include Visa®, Master Card™, banking institutions (direct debit), etc. The payment module may be configured to provide a consumer with the ability to select (either automatically or manually) a particular payment account for making payment on a specified bill.

Additionally, the payment module provides the ability for billers to select a particular payment provider to pay a selected bill based on predetermined business logic (i.e. rules) or other criteria. A payment provider is a 3rd party service used by a biller to manage the physical payment mechanism relating to how a bill gets paid. Payment providers are used to facilitate payments authorized by consumers, and may perform such tasks as: credit card verification, money transfers, verification of funds, remittance info, etc. Examples of different payment providers include CheckFree or Columbus, Ohio, and PaymentNet of Mountain View, California.

In a specific embodiment of the present invention, the business logic used to select a particular payment provider may be implemented via rules in the same

manner by which the rules are implemented for the Presentation Server. Thus, an automated system may be set-up by the biller, if enabled, to select a particular payment provider to pay a specific bill, based upon a set of rules or other business logic which is customizable by the biller's system administrator.

The payment module may also be configured to provide a conditional auto-pay feature, whereby a bill is automatically paid if it is within a predetermined range as specified by the consumer. This may be implemented as an Action which is controlled by rules or other business logic. Additionally, the rendering system 200 of the present invention provides the ability for automatically generating notification of an automatically paid or non-paid bill.

Further, the payment module of the present invention may also be configured to provide the consumer with tracking information relating to the status of a payment authorized by the consumer to pay a selected bill.

FIGURE 20 shows a flow diagram of a specific embodiment for implementing the payment tracking procedure 2000 of the present invention. Procedure 2000 may be thought of as describing three states of payment tracking. The first state, payment of a particular bill is authorized by the user either manually (e.g. by clicking "PAY AMOUNT DUE") or automatically using business logic or rules. This is represented by block 2002 of FIGURE 20.

Once the payment has been authorized by the consumer, it enters a second state, whereby payment of the specified bill is initiated (2004). To initiate payment, biller information, including the billed amount, and consumer information, including customerIDentification and authorization of the billed amount will be forwarded to a selected payment provider. The particular payment provider selected may be explicitly chosen by the consumer, or automatically chosen using business logic or rules which are customizable by the user or consumer. Although not shown in FIGURE 20, notification may sent to the consumer advising the consumer that the payment has been initiated (i.e., sent to the consumer's selected payment provider). At this point, payment tracking enters a third state whereby confirmation or resolution information of the payment is awaited (2006).

Once the payment has been resolved, the outcome of the resolution is determined (2008). Typically, the outcome of the payment resolution will either be a denial (2012) or an authorization (2010) by the specified payment provider. This information is then forwarded to the consumer (2014). It will be appreciated by those

having skill in the art that, at any step along the way, the consumer may inquire as to the status of the payment. The rendering system 200 may respond with tracking information relating to the current state or status of the payment. If further information is available, this information may also be forwarded to user upon request.

Illustrative Example

It is intended that the following example will help clarify the dynamic page rendering technique of the present invention.

FIGURE 8 shows a flow diagram of an example of how a new session procedure 800 is implemented in accordance with a specific embodiment of the present invention. The session actually begins with a user or consumer accessing the Presentation Server 203 via the HTTP server 225. At 802, a new session (including a new session ID) is created within Presentation Server 203.

At 804, the log-on page is rendered by the Presentation Server. An example of a rendered log-on page is shown in FIGURE 5 of the drawings. After a user enters a user ID (502) and password (504), the userID and password data from the user are received at the Presentation Server at 806. At 808, a data pool object and Execution Cache for this session are created. Additionally, as shown in FIGURE 4 of the drawings and as described previously, the new session will also include an instance of the data accessors 421, and an instance of page rendering droplets 431. At 810, a SIGN-ON request is sent to the Execution Cache. An appropriate accessor handles this call to the Execution Cache.

At the execution cache, the input parameters (userID, Password) are used as keys to retrieve output data (e.g. customerID) from the Execution Cache. If the desired output data is not cached within execution cache 420 (Fig. 4) or if the cached data is not current, then the call request and input parameters are passed to the Integration Server to retrieve the desired data from the Legacy system. In the example of Fig. 8, it is assumed that the desired output data (for the Sign-on) is not cached within execution cache 420.

At 812, the Integration Server authenticates the user by accessing the Legacy system and comparing the user ID and password information obtained from the user with similar information obtained from the Legacy system. As will be described in greater detail below, when the Integration Server Authenticates the user, it accesses the Legacy database, passing to the Legacy system the appropriate input parameters.

The Legacy system will respond by providing appropriate output parameters which, in the example of FIGURE 8, include a customerID and password (PSSWD) related to this specific user. Each request to the Legacy system will result in different output data being provided from the system to the Integration Server. This output data is described within Meta file 240.

At 814, a determination is made whether the user is valid. If it is determined that the user is not valid, at 816 the session is terminated and at 818 an appropriate page is rendered.

If, however, it is determined that the user is valid, at 820, the customerID data received from the Integration Server is stored into the data pool instance 426 for that session. At 822, a determination is made whether the SIGN-ON request output parameters are marked as volatile in the Meta file 240 (FIGURE 4). If these output request parameters are marked as volatile, then they are not cached, and the procedure continues at 826, wherein the next page (e.g. Account Summary Page, FIGURE 6) is rendered. If, however, the output parameters for the SIGN-ON request are not marked as volatile in the Meta file, then the output parameters are cached within Execution Cache 420 (FIGURE 4) at 824, and the next page is then rendered at 826.

FIGURE 9 shows a flow diagram for rendering an Account Summary Page in accordance with a specific embodiment of the present invention. An example of an Account Summary Page is shown in FIGURE 6 of the drawings.

Returning to FIGURE 9, at 902, the session ID (which was created in 802 of FIGURE 8) is obtained either from a cookie placed in the client's browser (if supported), a URL, or internally.

At 904, a determination is made whether the desired template for the Account Summary Page is cached within the Presentation Server 203. If the template is not cached within the Presentation Server, at 908 it is retrieved from the template directory 222. The retrieved template is then cached in the Presentation Server at 910. In a specific embodiment, the cached template is stored in Page Cache 457 of the Presentation Server (Fig. 4).

At 912, the Account Summary Page is then rendered as shown, for example, in FIGURE 6. The page rendering process is described in greater detail with respect to Fig. 10 of the drawings.

Returning to 904, if it is determined that the desired template is cached within the Presentation Server, than at 906 a second determination is made as to whether the

cached template is the most current version. If it is not, flow is continues at 908, as described previously. If, however, the cached template within the Presentation Server is the most current version of the account summary template, it is rendered at 912.

It will be appreciated by those having skill in the art that, once the template has been cached within the Presentation Server, most of the code relating to this template has already been compiled. Accordingly, the template is able to be rendered much more quickly by the Presentation Server after it has been rendered at least once, since the compiled code is already stored within the Presentation Server. In a specific embodiment, at least a portion of the compiled code is stored within the Page Compilation block 458 of the Presentation Server (Fig. 4).

FIGURE 10 shows a flow diagram of a specific embodiment for implementing a Render Page procedure such as that identified by block 912 in FIGURE 9.

The Render Page procedure 1000 is implemented by the Presentation Server 203 using a selected templates from the Template directory 222 (FIGURE 4). As discussed previously, each page template includes at least one static portion and at least one customized portion. At 1001, a first static segment or portion of the template page is rendered. At 1002, a determination is made as to whether there are any remaining segments to be rendered in the page template. If there are no remaining segments to be rendered (i.e if the entire page has been rendered), at 1060, flow control returns to a previous procedure.

If, however, it is determined that there are further segments to be rendered from the page template, at 1003, a determination is made as to whether the next segment contains a custom tag. If the next segment to be rendered does not contain a custom tag, it will be processed as a static segment at block 1001.

If a custom tag is encountered in the next segment, the particular type of custom tag is determined and processed accordingly, as described below.

There are a number of various types of custom tags which may be encountered when processing a page template, such as, for example, "value of" tags, "set value" tags, droplet tags, etc.

A "value of" custom tag represents information which is to be retrieved from the Legacy database. An example of a "value of" tag is shown in Fig. 11B. A "set value" custom tag represents information which is to be stored or modified in the Legacy database.

If the encountered custom tag is a "set value" tag, then at 1032, the "set value" tag is processed in a manner similar to the processing of a "value of" tag (described in detail below.

If the custom tag is a droplet, at 1042, the droplet is processed by looking up the name of the droplet in Nucleus 454 (Fig. 4), executing code for the particular droplet within instance 431, and then replacing the droplet tag with the results of the executed code.

Custom tags which are not "value of" tags, "set value" tags, or droplets are processed accordingly as shown at 1050.

If the custom tag encountered is a "value of" tag, then, at 1006, an appropriate data accessor instance is called to process the custom tag.

At 1007, the Meta file 240 is consulted to identify any input parameters related to this custom tag which are to be retrieved from the data pool 426. The custom tag will typically be associated with a particular request which is to be made upon the Integration Server to access data from the Legacy system. Each request typically has one or more associated input parameters. If the desired input parameters are not included within the data pool, they were most likely retrieved either from the consumer (via a form, for example), or from the Integration Server.

At 1009, the request input parameter(s) are retrieved from the data pool 426 (FIGURE 4). At 1011, the request, along with its associated input parameter(s) are sent to the Execution Cache 420.

At 1013, a determination is made whether the request (associated with the custom tag encountered at 1003) is marked as volatile in Meta file 240. If the request is not marked as volatile, at 1015, the Execution Cache 420 is checked to see if the anticipated results (for the associated request and input parameters) have been previously stored within the Execution Cache.

If the anticipated results of the request have not been stored in the Execution Cache 420, or if the request is marked as volatile in the Metal file, the request, along with its input parameters, are sent to the Integration Server to be processed at 1017. At 1020, the Integration Server 201 processes the request and returns the results of the request to the Presentation Server. At 1021, a selected portion of the request results are stored in data pool 426. The particular data which is to be stored in the data pool is determined by consulting the information contained within Meta file 240. An

example of the technique used for determining which data to store in the data pool is described previously with respect to Figs. 16 and 17.

At 1022, a determination is made whether the parameters associated with the processed request is marked as volatile in the Meta file. If the request parameters are not marked as volatile in the Meta file, at 1024, the results returned from the Integration Server (i.e., the output parameters associated with the processed request) are cached in the Execution Cache 420.

Once the desired information relating to the custom tag has been acquired at the Presentation Server (either by retrieving this information from the Execution Cache 420 or by retrieving this information from the Integration Server 201), the desired data is formatted, if formatting is specified for this data (1018)

At 1019, the value of the custom tag is rendered by replacing the custom tag with the retrieved results or output parameters. The procedure then returns to block 1002 to see if any additional segments remain to be rendered in the template.

FIGURE 12 shows a flow diagram of an Integration Server Process Request procedure 1200 which is called, for example, in block 1020 of FIGURE 10.

FIGURE 13 shows a schematic block diagram of a communication path of a specific embodiment of the rendering system of the present invention. This figure may be used in conjunction with the explanation of the flow diagram of FIGURE 12.

As shown in FIGURE 13, the Presentation Server sends a request along with input parameters to the Integration Server 201. In a specific embodiment of the present invention, the request is transmitted using an XML protocol. This request is received by the Integration Server as an "execute" command. The data in the XML request identifies the name of the request along with the associated input parameters for that request (1201, Fig. 12).

At 1203, the Integration Server uses the request name and associated input parameters to identify a suitable top level Plug-in for handling the request. The identification of the top level Plug-in is based, at least in part, upon the request name and Plug-in Meta information contained within the Request/Plug-in Mapper database 273 (Fig. 2).

Once the suitable top level Plug-in (1308, Fig. 13) has been identified, the Integration Server passes the request along with the associated input parameters to the identified Plug-in at 1205. The identified Plug-in is selected from one of the plurality of Plug-ins 202 shown in FIGURE 2 of the drawings.

At 1207, the top level Plug-in Roots the request to an appropriate Root Action (1310, Fig. 13). At 1209, the Root Action executes appropriate other Action(s) as required. These other Actions are represented in FIGURE 13 by block 1312.

At 1211, each Action within block 1312 may call one or more DOMs (1314, Fig. 13), as required. In a specific embodiment of the present invention, an XML protocol is used for communicating between the Actions and the DOMs.

At 1213, each DOM accesses the Legacy database (210, Fig. 13) in accordance with the call request from the calling Action, and converts any retrieved Legacy data into a format which may be read and manipulated by the calling Action. In a specific embodiment of the present invention, each DOM converts retrieved Legacy data to an XML format.

At 1215, the converted Legacy data is passed from the DOM(s) to the appropriate calling Action(s). At 1217, the Action(s) manipulate the retrieved data (if required) and pass the data to the top level Plug-in 1308.

At 1219, the top level interface or Plug-in 1308 further manipulates the retrieved data (if required), and passes the data to the Presentation Server via the Integration Server interface. The Presentation Server then renders the data for display to the user.

FIGURES 15A and FIGURES15B provide an illustrative example of how an Authenticate SIGN-ON request is handled by the rendering system 200 of the present invention. The Authenticate SIGN-ON request may be issued from the Presentation Server and processed by the Integration Server, as shown, for example, in blocks 810 and 812 of the flow diagram of FIGURE 8. The flow diagram of FIGURE 15B will now be described with reference to the schematic block diagram of FIGURE 15A.

Referring to FIGURE 15A, the Presentation Server 203 sends a SIGN-ON request along with associated input parameters (e.g., user ID, password) to the Integration Server 201. The Integration Server receives the SIGN-ON request along with the user ID and password as input parameters (1520, FIGURE 15B). The Integration Server then identifies the Authentication Plug-in as the appropriate Plug-in for handling this request, and passes the request name and input parameters to the Authentication Plug-in (1522).

The Authentication Plug-in is represented in FIGURE 15A by block 1502.

At 1524, the Authentication Plug-in passes the input parameters to Root SIGN-ON Action 1504. At 1526, the Root SIGN-ON Action calls a generic SIGN-

ON Action 1506, passing the user ID and password as input parameters. The generic SIGN-ON Action includes a cache 1506A for caching input and/or output parameters associated with the SIGN-ON request.

At 1528, the generic SIGN-ON Action calls an "Access User" DOM 1508 to access the Legacy database, passing the user ID as an input parameter for the request.

At 1530, DOM 1508 passes the user ID information to the Legacy database, and retrieves a plurality of output parameters, which include customerIDentification data (CUST ID), password data (PSSWD), and other data.

At 1532, the DOM 1508 returns the retrieved Legacy data (CUST ID, PSSWD, and other data) to the generic SIGN-ON Action 1506.

At 1534, the generic SIGN-ON Action compares the consumer password (password) to the retrieved Legacy password (PSSWD). If the passwords are not the same, at 1550, an invalid password error is passed to the Root SIGN-ON Action 1504. At 1552, the Root SIGN-ON Action passes the invalid password error to the Authentication Plug-in 1502, and, at 1554, the Authentication Plug-in returns an invalid password error to the Integration Server.

If, however, it is determined by the generic SIGN-ON Action that the passwords are the same, then at 1538, the customerID information and other data retrieved from the Legacy system are passed from the generic SIGN-ON Action to the Root SIGN-ON Action.

The Root SIGN-ON Action then calls other Actions as appropriate. This is represented in FIGURE 15A by block 1512. Examples of other Actions may include determining when the consumer last logged on to the system.

At 1542, the Root SIGN-ON Action returns the customerID information and other data to the Authentication Plug-in. At 1544, the Authentication Plug-in returns the customerID and other data to the Integration Server. At this point, the Authentication procedure is completed, and flow control is returned to a previous procedure.

In a specific embodiment of the present invention, the rendering system 200 is designed or configured to provide a biller, creditor, or other institution with the ability to provide electronic bill presentment capabilities to their respective consumers via the Internet.

FIGURE 7 shows an example of a graphical user interface which has been rendered in accordance with the technique of the present invention The graphical

user interface of FIGURE 7 shows a web page displaying specific billing information associated with a particular user. More specifically, the web page presents information relating to long distance calls itemized in the user's telephone bill.

Table 700 of FIGURE 7 shows an itemized list of the long distance calls made by the user during a specific time period. Each itemized line (e.g., 702, 704, 706, etc.) within Table 700 includes customized data retrieved from the Legacy computing system, which in this case is the Central Telecom computing system. Table 700 is generated using a reusable droplet such as, for example, the ForEach droplet described previously.

Table 700 includes a plurality of rows and columns. Each of the columns relates to a specific type of information (e.g., date, time, location, etc.) of a long distance call. The user is able to sort the information presented in Table 700 by clicking on a particular header (e.g., 712A, 712B, 712C, etc.).

As shown in FIGURE 7, information within Table 700 is arranged by date, in descending order (indicated by the downward arrow in field 712A). A ForEach droplet is used by the Presentation Server 203 to present information to the user in a sorted format. The ForEach droplet also includes code for reversing the sorted order upon receiving a request by the user. In the example of FIGURE 7, a user can reverse the sort order (i.e., sort by date in ascending order) by simply clicking on the date header 712A. Each time the user clicks on the date header, the sorting order is switched from ascending to descending order or vice versa.

Each column header in table 700 represents a sorting key which, when selected by the user, will result in the table being sorted according to the sorting key selected by the user. When a specific header is selected by the user, a request is sent to the Presentation Server to display the table information according to the selected sorting key. In this way, an embodiment of the present invention may be implemented to allow a user to sort the information within Table 700 according to any of the headers by simply clicking on the desired header within row 712. Alternatively, it may be desirable to enable only a portion of the table headers to be used as sorting keys. It will be appreciated by those skilled in the relevant art that the sorting key may also be used for sorting rows as well as columns.

As described previously, one advantageous feature of the rendering system of the present invention is the ability to provide the biller with means for implementing customer self-care, whereby the user is able to access desired information relating to

specific billing information over the Internet or other computer network, rather than having to telephone a customer service representative of the service provider. One example of customer self-care is the ability for the consumer to look up additional information about a transaction or vendor identified on the consumer's credit card bill.

Another example of customer self-care is the ability for the consumer to perform a reverse call look-up. To illustrate, let us suppose that the user does not recognize the telephone call identified by line 708 in FIGURE 7. In order to retrieve additional information relating to this call, the user may click on the unrecognized telephone number (366-533-3553). Each telephone number is configured as an object link. When a particular telephone number is selected, a request is sent to the Presentation Server to perform a reverse call look-up on this number The Presentation Server passes this request onto the Integration Server, which accesses the Legacy database to retrieve additional information relating to the selected telephone number. The Legacy system may respond, for example, by providing a name or other information related to the selected telephone number. This information, in turn, is presented to the consumer via the Integration and Presentation Servers. By allowing a consumer to perform various aspects of customer self-care, a service provider is able to save or conserve a tremendous amount of resources.

Because the rendering system 200 of the present invention is easily configurable and customizable, it is able to be integrated quickly and easily to most computer systems. Additionally, although not shown in the drawings, the rendering system 200 of the present invention is also highly scalable. Rendering system 200 may include multiple Presentation Servers and Integration Servers (not shown). Where multiple Presentation Servers exist, load balancing may be performed by a load manager (not shown), which periodically checks each Presentation Server load and announces this to the connection module (CM) of the HTTP server. The same basic approach is used between a Presentation Server and multiple Integration Servers.

E-mail Notification

FIGURES 18, 18A and 18B describe various features and methods related to the e-mail notification feature of the rendering system of the present invention. The various elements for implementing the e-mail notification feature of the present

invention are shown in FIGURE 18 of the drawings. These elements will now be described with respect to the flow diagram of the e-mail notification procedure 1800 of FIGURE 18B.

For purposes of explanation and simplification, the flow diagram of FIGURE 18B illustrates an example of a specific embodiment for providing e-mail notification to a consumer of a bill which has become available. However, it will be appreciated that the e-mail notification feature of the present invention may be extended to other uses relating to any event for which a particular consumer would like to be notified.

Referring first to Fig. 18B, when a bill for a particular consumer becomes available at a Legacy system (1850), the Legacy information relating to the bill is entered (1852) in the notification queue 1802 (Fig. 18). In a specific embodiment, the Legacy system is able to access the event notification queue 1802 using one or more DOMs. A schematic block representation of this communication path is shown, for example, in FIGURE 2.

As shown in FIGURE 2, any of the Legacy systems 210 may communicate with DOMs block 208. DOMs block 208 provides an interface between the Legacy system 210 and event notification block 277, which is represented in FIGURE 18 by the dashed block (277) enclosing the template directory 1804 and notification queue 1802.

As shown in FIGURE 18, the notification queue 1802 includes a plurality of entries 1805. An example of the information included within one entry 1805 is shown in FIGURE 18A of the drawings. In the example of FIGURE 18A, entry 1805 includes BillerID data, CustomerID data, AccountID data, and other data, including information relating to a respective TemplateID associated with a specific bill which has become available to the consumer. The specific field name headers in each entry 1805 are determined from the configuration information of block 224 (Fig. 2) during initialization. This latter feature provides for a generic e-mail notification system which may be customized according to the specific needs of the host or system administrator.

It is to be noted that entry 1805 of Fig. 18A is merely one example of the information included in a particular entry. Other entries may differ. However, it is preferable that each entry specify enough unique information to identify a specific consumer and a specific e-mail template which will be used for notifying the consumer of a particular event.

At periodic intervals, a timer within the mail rendering server 207 causes the mail rendering server to check the status of the notification queue 1802 via the Integration Server 201 (1854, FIGURE 18B). A determination is then made (1856) as to whether any new entries exist within the notification queue. If no new entries exist, the mail rendering server waits a predetermined amount of time before checking the status of the notification queue again for new entries (1857).

If new entries are found in the notification queue, a predetermined number of new queue entries is retrieved from the notification queue (1858). In a specific embodiment of the present invention, the predetermined number is specified in the configuration information of block 224 (FIGURE 2), and is configured upon start-up or initialization of the rendering system 200.

Once a new entry has been retrieved from the notification queue, the mail rendering server 207 uses the TemplateID information within that entry to look-up the associated template file name 1809 (Fig. 18) which will be used for rendering the e-mail notification message to the consumer (1860, Fig. 18B). The associated template file(s) are then retrieved from the template directory 222 (1862), and processed (1864) in a manner similar to the processing technique described previously with respect to the Presentation Server 203. Any data which can be rendered by the Presentation Server can be similarly rendered by the mail server 207. This also includes executing rules from rules block 220 in the same manner that rules are executed by this Presentation Server for rendering web pages. Once the e-mail template, including the customized tag information has been fully processed and rendered, the results of the processed or rendered template file, including the consumer specific data, is then sent (1866) to the consumer via the SMTP server 227.

Although several preferred embodiments of this invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to these precise embodiments, and at various changes and modifications may be effected therein by one skilled in the art without departing from the scope of spirit of the invention as defined in the appended claims.

IT IS CLAIMED

1. A rendering system for dynamically generating graphical user interface pages over a computer network using customized data from a backend computing system, the rendering system comprising:

a plurality of page templates, wherein at least some of the page templates include static information and include custom tags which represent customized data associated with data stored in said backend computing system;

an integration engine configured or designed to access specific information from the backend computing system; and

a presentation engine in communication with the integration engine, said presentation engine being configured or designed to render graphical user interface pages using the page templates such that when a first page corresponding to a first one of the page templates is accessed, the presentation engine utilizes the first page template associated with the first page to render the first page, wherein customized data to be included in the first page is obtained by accessing the integration engine.

2. The system of claim 1 wherein said integration engine is further configured or designed to provide and interface to enable a first computer system to access information from multiple remote data sources by normalizing data from said multiple remote data sources.

3. A rendering system as recited in claim 1 or 2, wherein the presentation engine includes:

an execution cache suitable for caching data retrieved from the backend computing system; and

a data pool for storing information used to make calls to the integration engine.

4. The system of claim 3 wherein said information stored by said data pool is inter-page information that is utilized by a first page and a second, subsequent graphical user interface page.

5. A rendering system as recited in any of the preceding claims further comprising:

a meta file which includes information describing properties of the backend computing system and data contained therein.

6. The rendering system of claim 5 wherein said meta file information includes:

a description of call requests which may be used to access data in said backend computing system:

a description of input parameters associated with each respective call: and

a description of output parameters associated with each call.

7. The rendering system of claim 5 or 6 wherein said presentation engine is further configured or designed to utilize said meta file information to access data from said backend computing system.

8. The rendering system of claim 7 wherein said presentation engine is further configured or designed to access said meta file to retrieve a description of input parameters related to a desired call request. and wherein the presentation engine is further configured or designed to pass said call request and associated input parameters to said integration engine in order to access data from said backend computing system.

9. The rendering system as recited in any of claims 5-8 wherein the presentation engine includes at least one generated accessor configured or designed to use information in the meta file to facilitate rendering of information.

10. The system of claim 9 wherein said information is obtained from either said integration engine or said execution cache.

11. A rendering system as recited in any of the preceding claims further comprising:

a graphical user interface server in communication with the presentation engine, the graphical user interface server being configured or designed to facilitate communications between the presentation engine and a remote user over the computer network.

12. A bill presentation system comprising a system for dynamically generating graphical user interface pages as recited in any of the preceding claims, wherein the bill presentation system is configured or designed to present on-line bills, the billing system further comprising:

a payment module configured or designed to facilitate payment of bills by a user, the payment module being in communication with the integration engine.

13. The billing system of claim 12 wherein said payment module is further configured or designed to provide said user with dynamic payment tracking information relating to a status of a payment which the user has authorized.

14. The billing system of claim 12-13 wherein said payment module is further configured or designed to provide a conditional auto-pay feature, whereby payment of a bill is automatically paid if an amount of said bill is within a predetermined range of values.

15. The billing system of any of claims 12-14 wherein said payment module is further configured or designed to provide a conditional auto-pay feature, whereby payment of a bill is automatically paid if specified criteria has been satisfied, said specified criteria being implemented via business logic which has been customized by the user.

16. The billing system of any of claims 12-15 wherein said payment module is further configured or designed to interface said user with a plurality of different payment providers.

17. The billing system any of claims 12-16 wherein said payment module is further configured or designed to allow a biller to dynamically select a particular payment provider from said plurality of payment providers to pay selected bills: and

wherein said payment module is further configured or designed to allow a consumer to dynamically select a particular payment account from a plurality of payment accounts to pay a selected bill.

18. The billing system of claim 16 wherein said payment module is further configured or designed to automatically select a particular payment provider from the plurality of payment providers to pay specific bills based on predetermined criteria specified by the biller.

19. A billing system as recited in any of the preceding claim further comprising a consolidation server in communication with the integration engine, the consolidation server being configured or designed to facilitate communications with an on-line bill consolidator that presents on-line billing information to the user.

20. A rendering system as recited in any of the preceding claims wherein the integration engine includes:

a integration server for communicating with the presentation engine;

a plurality of action objects configured or designed to implement business logic; and

a plurality of domain objects that are accessible by the action objects, the domain objects being configured or designed to access customized information from the backend computing system and present said accessed information to the integration server.

21. A method of targeting advertising information to a user over a computer network, the method comprising:

rendering an on-line itemized bill having at least one billed item;

consulting a set of rules to select a particular advertisement to display to said user, said set of rules being based at least in part upon properties of said bill, and

rendering the particular advertisement in combination with the on-line itemized bill.

22 The method of claim 21 further including:

accessing billing information related to said user from a backend computing system: and

using at least a portion of said accessed billing information to select the particular advertisement to display to the user

23. The method of claim 21 or 22 wherein said billing information is different from profile information relating to said user.

24. A method for dynamically generating graphical user interface information over a computer network using customized information from a backend computing system. the method comprising:

rendering at least a portion of an on-line itemized bill for presentation to a user. said bill having at least one billed item:

consulting a set of rules to select a particular advertisement to display to said user, said set of rules being based at least in part upon information relating to said bill, and

rendering the selected advertisement in combination with the on-line itemized bill.

25. The method of claim 24 further including:

accessing billing information related to said user from a backend computing system; and

using at least a portion of said accessed billing information to select the particular advertisement to display to the user:

wherein said billing information is different from profile information relating to said user.

26. A method for dynamically generating graphical user interface pages over a computer network using customized information from a backend computing system. the method comprising:

receiving a request at a first server for selected information to be presented to a remote user:

making a call request to an integration server for accessing said selected information, said call request being issued by said first server utilizing a thin interface protocol;

using said integration server to access said selected information from a backend computing system, and retrieving output information relating to said call request;

passing said output information to said requesting server; and

rendering said output information for presentation to said remote user.

27. The method of claim 26 wherein said call request is implemented by passing a single parameter to said integration server, said single parameter including tagged, self-described information relating to said call request.

28. The method of claim 27 wherein said self-described information includes a plurality of input parameters required by the integration server for making said call request.

29. The method as recited in any of the claims 26-28 further including consulting meta information describing said call request to determine the information to be included within said single parameter.

30. The method as recited in any of the claims 26-29 further including consulting a data pool for retrieving input parameter data related to making said call request.

31. The method as recited in any of the claims 26-30 further including caching said output information retrieved by the integration server in an execution cache.

32. The method of claim claims 31 further including retrieving said output information related to said selected information from said execution cache, and presenting said output information to said remote user.

33. The method of claims 31 or 32 wherein said output information is retrieved from the execution cache and presented to said remote user without making a call to the integration server to retrieve said output information.

34. The method as recited in any of the claims 26-33 further including caching at least a portion of said output information in said data pool.

35. The method of claim 34 further including consulting meta information to determine said portion of said output information to store in said data pool.

36. A computer program product for dynamically generating graphical user interface pages over a computer network using customized information from a backend computing system. the computer program product comprising:

a computer usable medium having computer readable code embodied therein. the computer readable code comprising:

computer code for receiving a request at a first server for selected information to be presented to a remote user;

computer code for making a call request to an integration server for accessing said selected information. said call request being issued by said first server utilizing a thin interface protocol;

computer code for using said integration server to access said selected information from a backend computing system. and retrieving output information relating to said call request:

computer code for passing said output information to said requesting server; and

computer code for rendering said output information for presentation to said remote user.

37. An integration engine for facilitating communications between a legacy computing system and a graphical user interface rendering system to facilitate the inclusion of customized data from the legacy computing system in at least one graphical user interface page rendered by the rendering system. the integration engine comprising:

a integration server for communicating with the rendering system:

a plurality of action objects configured or designed to implement business logic: and

a plurality of domain objects that are accessible by the action objects, the domain objects being configured or designed to interface with the legacy computing system in order to access said customized data, and present said accessed data to said integration server.

38. A mail rendering system comprising the integration engine of claim 37, the mail rendering system further comprising a mail rendering server in communication with the integration engine for dynamically generating at least one electronic mail message to a particular customer upon the occurrence of at least one specific event.

39. The mail rendering system of claim 38 wherein said mail rendering server is configured or designed to dynamically generate a content of said electronic mail message based upon billing information related to the customer.

40. The mail rendering system of claims 38 or 39 further including an event notification data structure in communication with said integration engine for registering an occurrence of said at least one specific event.

41. The integration engine of claim 37 wherein said integration server is implemented using a thin server interface.

42. The integration engine of claim 37 or 41 further including a plurality of plug-in objects configured to communicate with said integration server: and

wherein said action objects and said domain objects are each implemented as plug-in objects.

43. The integration engine of claim 42 wherein one of said plug-in objects is configured as a payment module for facilitating payment of bills by a user.

44. The integration engine of claims 42 or 43 wherein:

at least one action object is configured to communicate with at least one of said plug-in objects; and

at least one of said domain objects is configured to communicate with at least one of said action objects.

45. The integration engine as recited in any of the claims 37-44 wherein at least one of said action objects is configured as a plurality of nested action objects.

46. A method for facilitating communications between a legacy computing system and a graphical user interface system to facilitate the inclusion of customized information from the legacy computing system in at least one graphical user interface page rendered by the rendering system, the method comprising:

receiving a first request for accessing desired information from the legacy computing system;

calling at least one plug-in object for servicing said first request;

calling a first action object for handling at least a first portion of said first request; and

calling a first domain object for accessing at least a first portion of said desired information from the legacy computing system.

47 The method of claim 46 further including receiving a single input parameter along with the first request, the single input parameter including tagged data describing a name of a particular call request to be implemented, and describing at least one input parameter associated with said particular call request.

48. The method of claims 46 or 47 wherein the single input parameter implemented as a single data string type.

49. The method as recited in any of the claims 46-48 further including using the tagged descriptions of the single input parameter to identify the name of the particular call request to be implemented, and to identify any input parameters associated with said particular call request

50. The method as recited in any of the claims 46-49 further including calling a second action object for handling at least a second portion of said first request.

51. The method of claim 50 wherein the second action object is called by said plug-in object.

52. The method of claim 50 wherein the second action object is called by said first action object.

53. The method as recited in any of the claims 46-52 further including calling a second domain object for accessing at least a second portion of said desired information from the legacy computing system.

54. The as recited in any of the claims 46-53 further including:

passing at least one first input parameter from said first plug-in object to said first action object;

passing at least one second input parameter from said action object to said domain object; and

using said at least one second input parameter to access at least a portion of said desired information from the legacy computing system.

55. A computer program product for facilitating communications between a legacy computing system and a graphical user interface system to facilitate the inclusion of customized information from the legacy computing system in at least one graphical user interface page rendered by the rendering system. the computer program product comprising:

a computer usable medium having computer readable code embodied therein. the computer readable code comprising:

computer code for receiving a first request for accessing desired information from the legacy computing system;

computer code for calling at least one plug-in object for servicing said first request;

computer code for calling a first action object for handling at least a first portion of said first request: and

computer code for calling a first domain object for accessing at least a first portion of said desired information from the legacy computing system.

56. A presentation engine for dynamically generating graphical user interface information over a computer network using customized data from a database of a backend computing system. the presentation engine comprising:

a presentation server configured or designed to render graphical user interface pages using page templates. the presentation server being further configured or designed such that when a first one of the page templates corresponding to a first page is accessed. the presentation server utilizes the first page template to render at least a portion of the first page. said first page including at least one customized tag corresponding to data derived the backend computing system;

an execution cache configured or designed to cache data retrieved from the backend computing system; and

a data pool for storing information used to make calls to access the customized data.

57. The presentation engine of claim 56 wherein said information in said data pool includes input parameters which are utilized for making calls to retrieve the customized data from the backend computing system.

58. The presentation engine as recited in any of the claims 56-57 wherein said information in said data pool includes input parameters which are utilized for making calls to retrieve the customized data from the execution cache.

59. A presentation engine as recited in any of the claims 56-58 further comprising a plurality of reusable droplets configured or designed to manipulate the presentation of customized data rendered by the presentation engine based upon user actions when viewing the customized data.

60. A presentation engine as recited in claim 59 wherein said plurality of droplets includes a reusable sorting droplet configured to be applied against tabular data to provide a sorted HTML table of data.

61. A presentation engine as recited in claim 60 wherein said sorting droplet is further configured or designed to sort said tabular data according to a sorting key selected by a user.

62. A presentation engine as recited in any of the claims 56-61 further comprising:

a page template directory having a plurality of page templates, wherein at least some of the page templates include tags which represent customized data to be retrieved from the backend computing system database; and

a meta file configured or designed to describe properties of the backend computer system database:

wherein the presentation server is configured or designed to utilize information from said meta file to access said customized data from said backend computer system database.

63 The presentation engine of claim 62 wherein said meta file information includes:

a description of call requests which may be used to access data in said backend computing system;

a description of input parameters associated with each respective call request; and

a description of output parameters associated with each call request

64. The presentation engine of claims 62 or 63 wherein said meta file information further includes a description of status codes associated with each call request.

65. The presentation engine as recited in any of the claims 62-64 wherein said presentation server is further configured or designed to access said meta file to retrieve a description of input parameters related to a desired call request, and wherein

the presentation server is further configured or designed to access said customized data using said call request and associated input parameters.

66. The presentation engine as recited in any of the claims 62-65 wherein the presentation server further includes at least one generated accessor configured or designed to use information in the meta file to render information obtained from said backend computer system database

67. A method for dynamically generating graphical user interface information over a computer network using customized information from a backend computing system, the method comprising:

receiving a request for selected information to be presented to a remote user:

rendering a selected page template for presentation to said remote user in response to said request, said page template including at least one custom tag representing customized data derived from the backend computing system:

consulting a meta file for determining at least one input parameter used for accessing said customized data; and

retrieving said at least one input parameter from a local data pool.

68. The method of claim 67 further including using said retrieved input parameter to poll a local cache for a valid copy of said customized data.

69. The method of claims 67 or 68 further including:

using said retrieved input parameter to access said customized data from said backend computing system in response to a determination that a valid copy of said customized data is not stored within said local cache; and

using said retrieved input parameter to access said customized data from said local cache in response to a determination that a valid copy of said customized data is stored within said local cache.

70. The method as recited in any of the claims 67-69 further including:

retrieving output data related to said customized data from the backend computing system:

if specified, caching at least a portion of said output data in a local cache; and

consulting said meta file to determine whether any portion of said output data should be stored within said data pool.

71. The method as recited in any of the claims 67-70 further including storing at least a portion of said output data in said data pool in response to a determination that said portion of output data may be used as input data to a subsequent request for accessing other customized data derived from said backend computing system.

72. A computer program product for dynamically generating graphical user interface information over a computer network using customized information from a backend computing system, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for receiving a request for selected information to be presented to a remote user;

computer code for rendering a selected page template for presentation to said remote user in response to said request, said page template including at least one custom tag representing customized data derived from the backend computing system;

computer code for consulting a meta file for determining at least one input parameter associated with accessing said customized data; and

computer code for retrieving said at least one input parameter from a local data pool.

73. A method of facilitating automated customer self care in an on-line billing environment implemented over a computer network, the method comprising:

rendering an on-line itemized bill to a consumer, said on-line bill having at least one billed item that includes an associated billed item link;

when the billed item link is selected by the consumer, accessing additional information about said billed item to help identify or clarify said billed item; and

providing said additional information to said consumer.

74. The method of claim 73 wherein said rendering includes rendering said on-line bill to said consumer over the Internet; and wherein said accessing includes accessing said additional information from said backend computing system over the Internet.

75. A method of facilitating automated customer self care as recited in claims 73 or 74 wherein the bill is a phone bill, and wherein said accessing includes performing a reverse call lookup on at least one phone number selected by said consumer.

76. A method of facilitating automated customer self care as recited in any of the claims 73-75 wherein the bill is a credit card bill, the billed item includes a transaction record, and wherein said accessing includes performing a lookup which provides additional information about said transaction record.

77. The method of claim 76 wherein said transaction record includes a vendor identification, and wherein said accessing includes performing a lookup which provides additional information about said identified vendor.

78. A computer program product for facilitating automated customer self care in an on-line billing environment implemented over a computer network, the computer program product comprising:

a computer usable medium having computer readable code embodied therein, the computer readable code comprising:

computer code for rendering an on-line itemized bill to a consumer, said on-line bill having at least one billed item that includes an associated billed item link;

computer code for when the billed item link is selected by the consumer, accessing additional information about said billed item to help identify or clarify said billed item; and

computer code for providing said additional information to said consumer.

79 A computer program product comprising a computer readable medium including code configured to implement the method of any of the preceeding claims

80. The computer program product of claim 79 wherein the computer readable medium is a data signal embodied in a carrier wave.

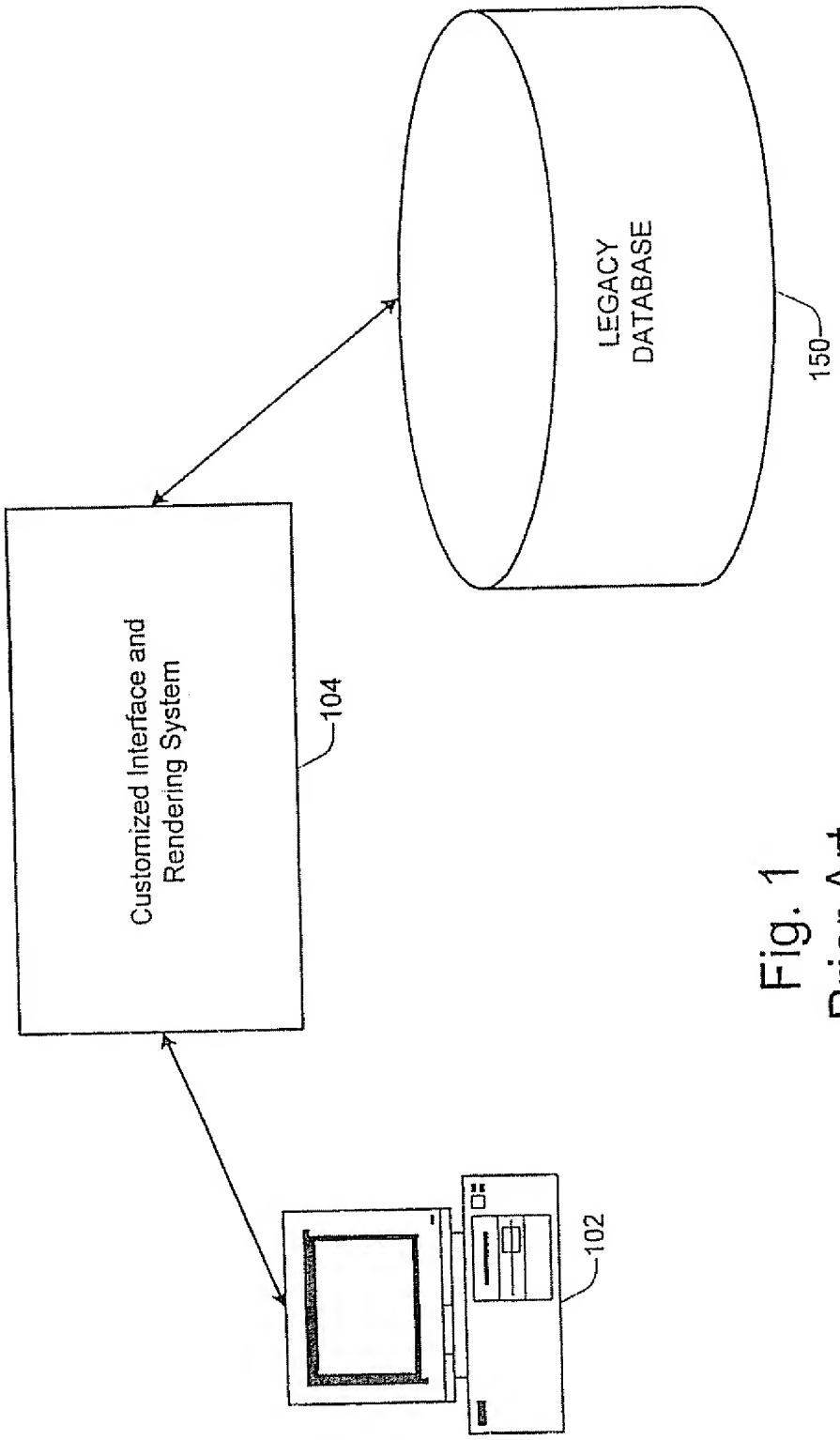


Fig. 1
Prior Art

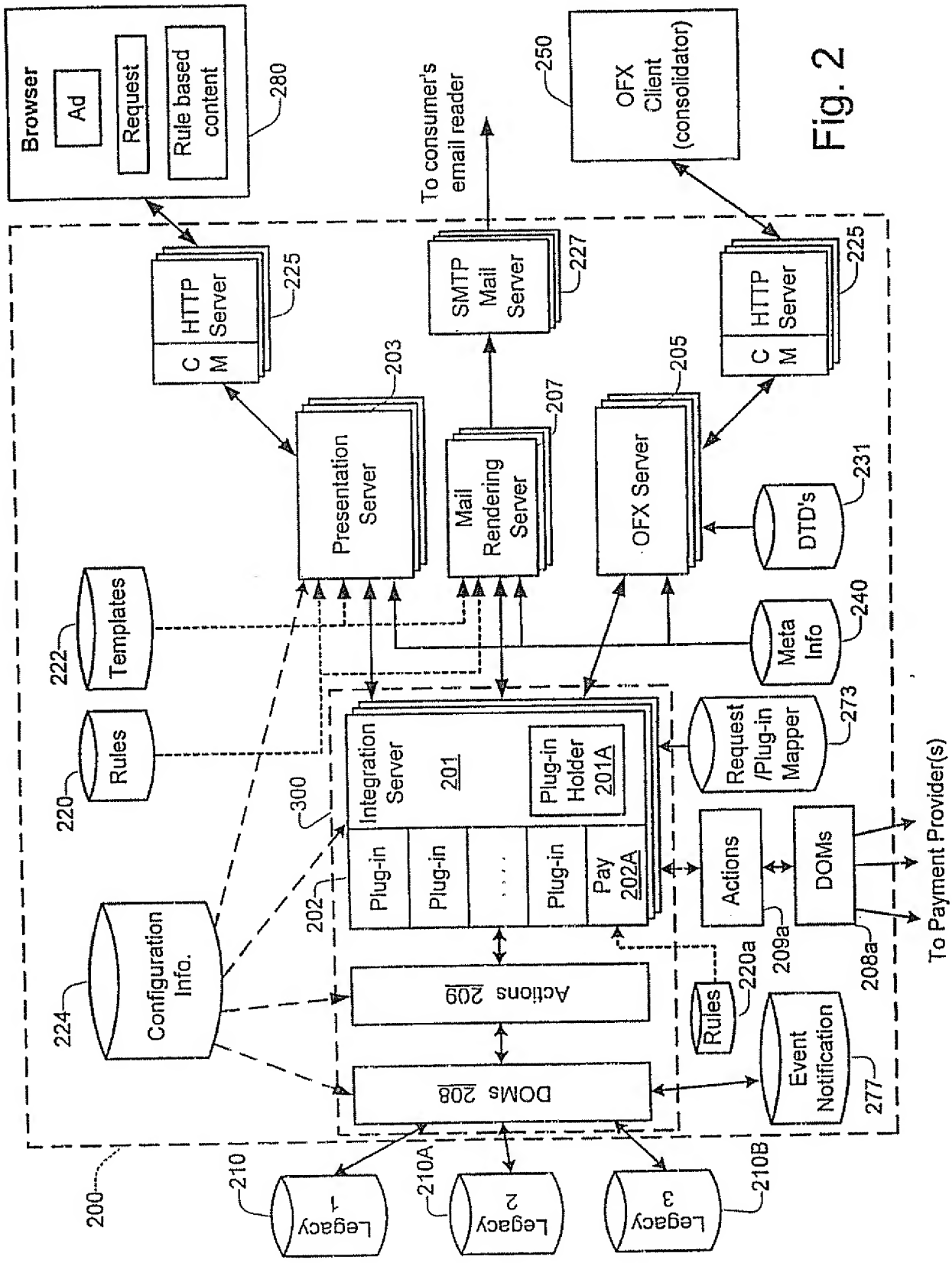

Browser
Ad
Request
Rule based content
280
To consumer's email reader
250
OFX Client (consolidator)
HTTP Server
C M
225
SMTP Mail Server
227
HTTP Server
C M
225
203
Presentation Server
207
Mail Rendering Server
205
OFX Server
231
DTD's
Templates
222
Meta Info
240
Rules
220
Integration Server 201
Plug-in Holder 201A
300
Request /Plug-in Mapper
273
Plug-in
Plug-in
Plug-in
Pay 202A
202
Actions
DOMs
209a
208a
220a
Configuration Info.
224
Actions 209
DOMs 208
Rules
Event Notification
277
To Payment Provider(s)
200
210
210A
210B
Legacy 1
Legacy 2
Legacy 3

Fig. 2

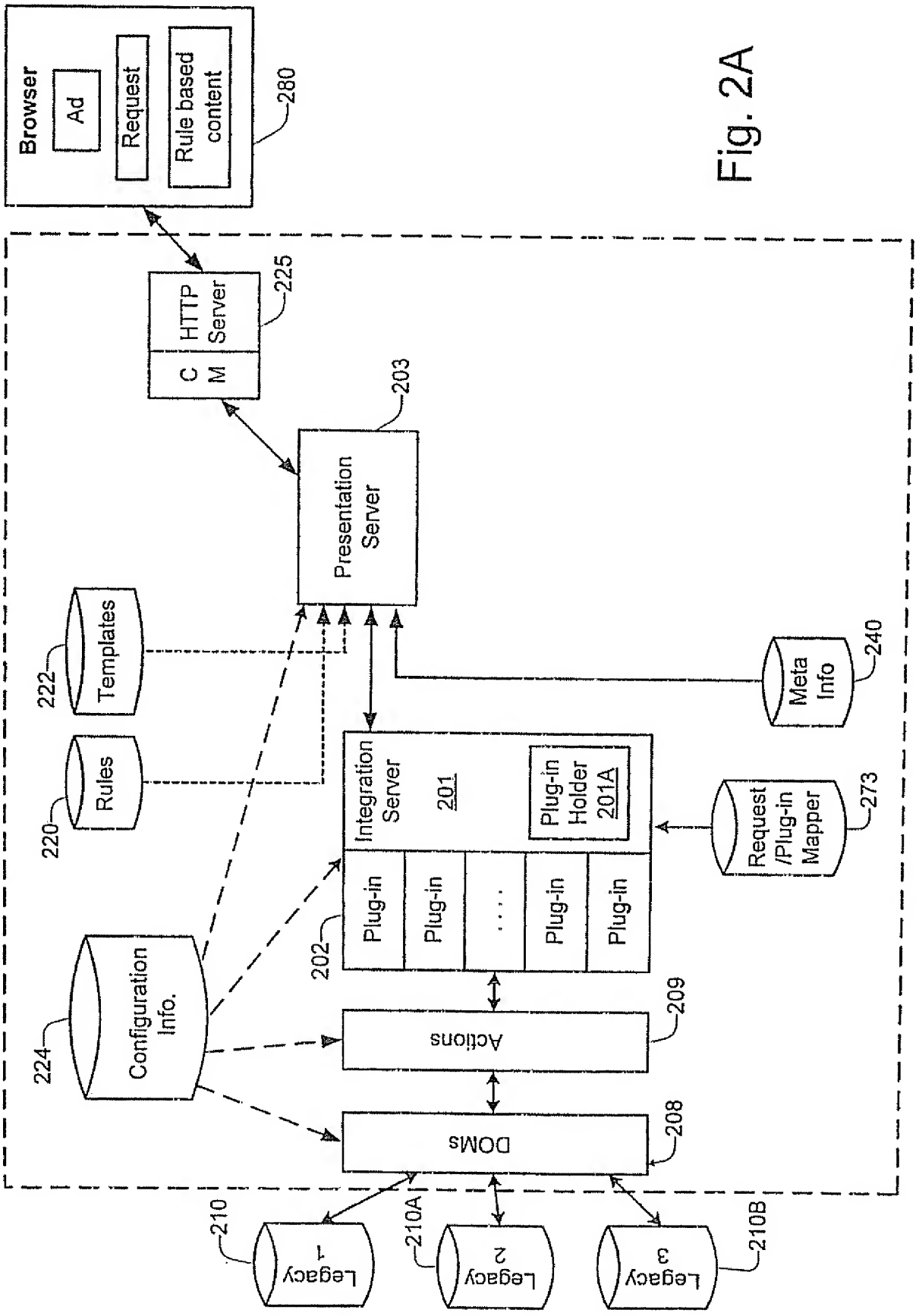

Browser
Ad
Request
Rule based content
280
HTTP Server
C M
225
Presentation Server
203
Templates
222
Rules
220
Meta Info
240
Integration Server 201
Plug-in Holder 201A
Request /Plug-in Mapper
273
Plug-in
Plug-in
. . . .
Plug-in
Plug-in
202
Configuration Info.
224
Actions
209
DOMs
208
Legacy 1
210
Legacy 2
210A
Legacy 3
210B

Fig. 2A

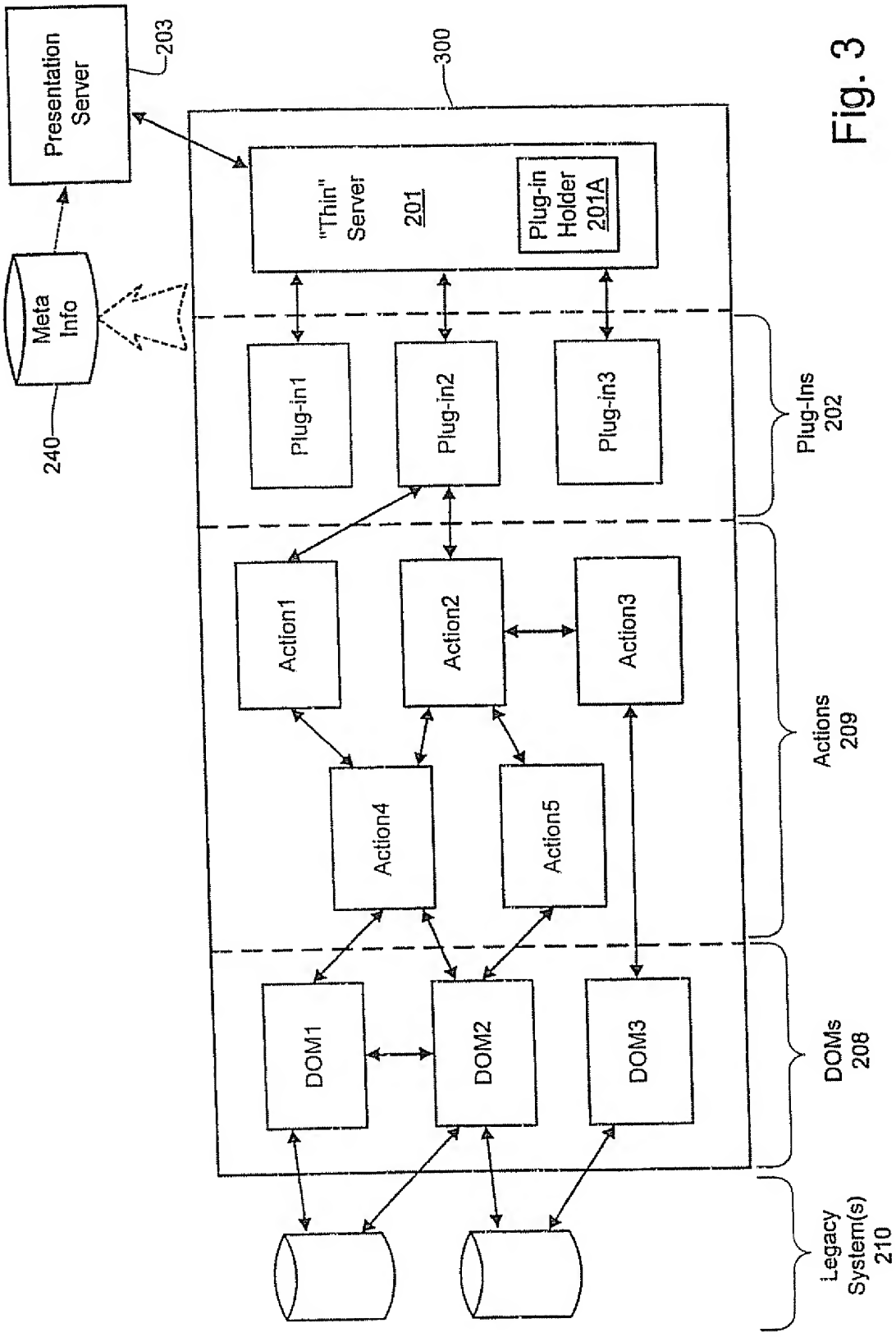

Presentation Server
203
300
"Thin" Server 201
Plug-in Holder 201A
Meta Info
240
Plug-in1
Plug-in2
Plug-in3
Plug-Ins 202
Action1
Action2
Action3
Action4
Action5
Actions 209
DOM1
DOM2
DOM3
DOMs 208
Legacy System(s) 210

Fig. 3

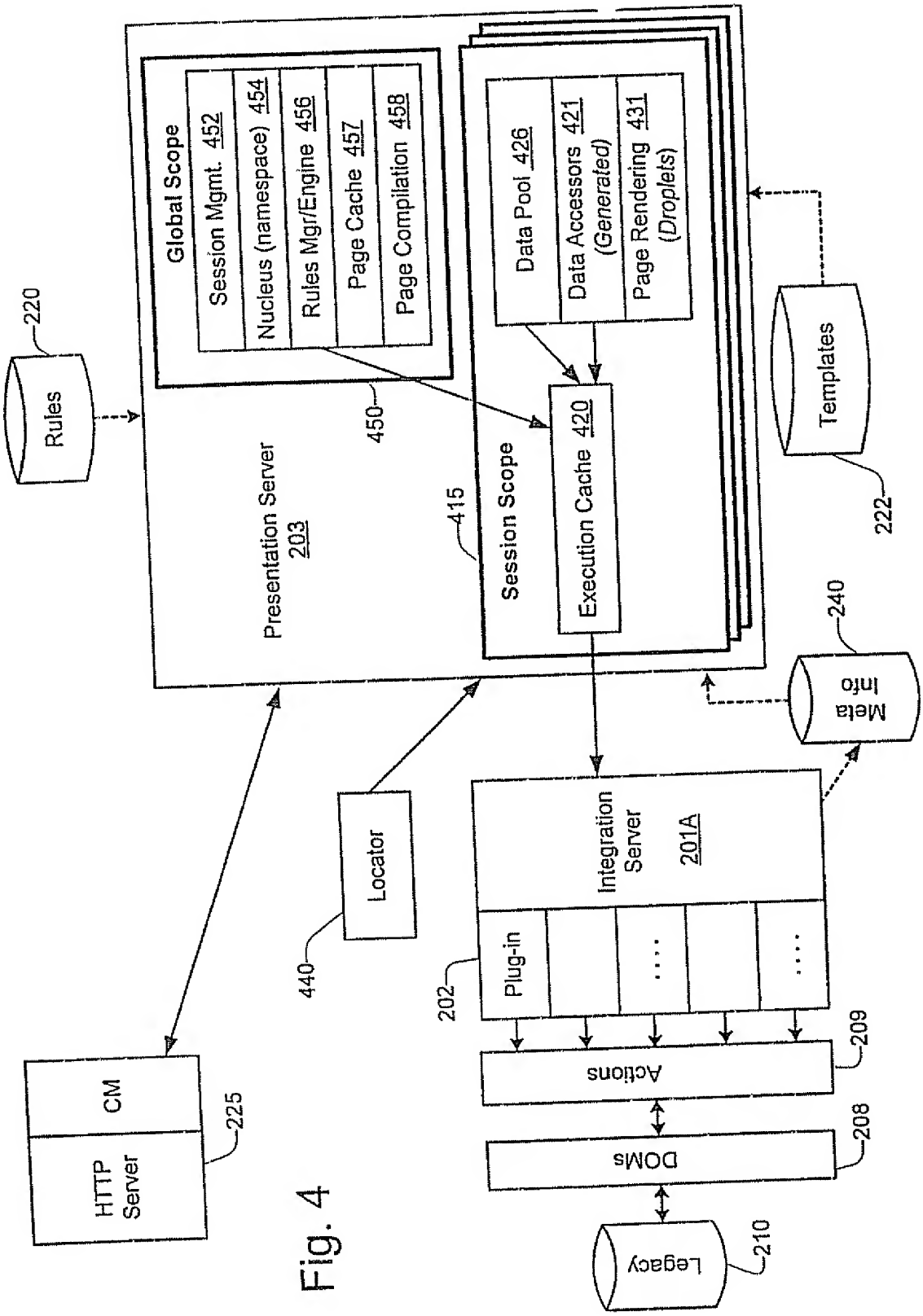

Global Scope
Session Mgmt. 452
Nucleus (namespace) 454
Rules Mgr/Engine 456
Page Cache 457
Page Compilation 458
Data Pool 426
Data Accessors 421 (Generated)
Page Rendering 431 (Droplets)
Execution Cache 420
Session Scope
415
450
Presentation Server 203
Rules
220
Templates
222
Meta Info
240
Integration Server 201A
Plug-in
202
Actions
209
DOMs
208
Legacy
210
Locator
440
HTTP Server
CM
225

Fig. 4

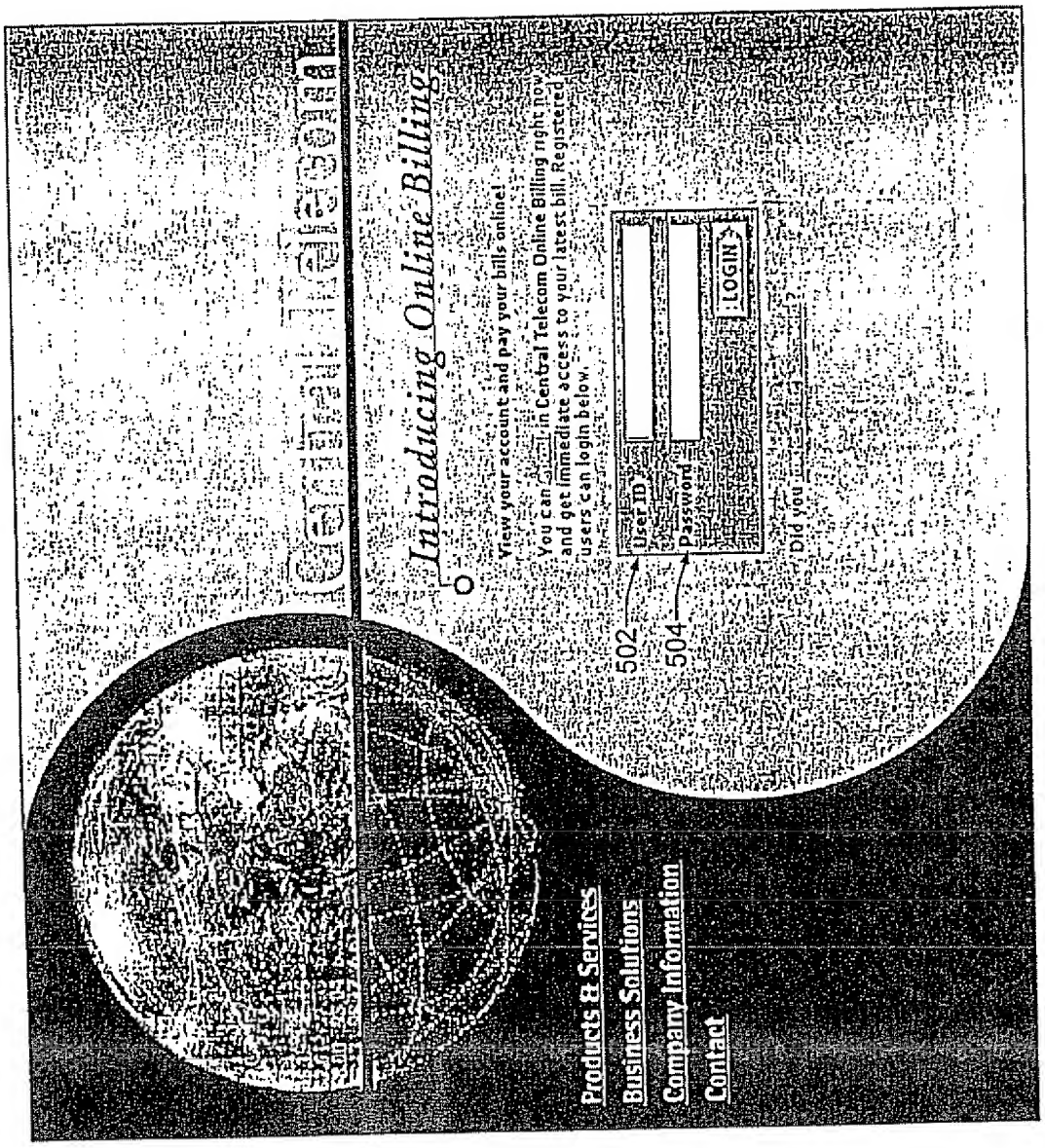
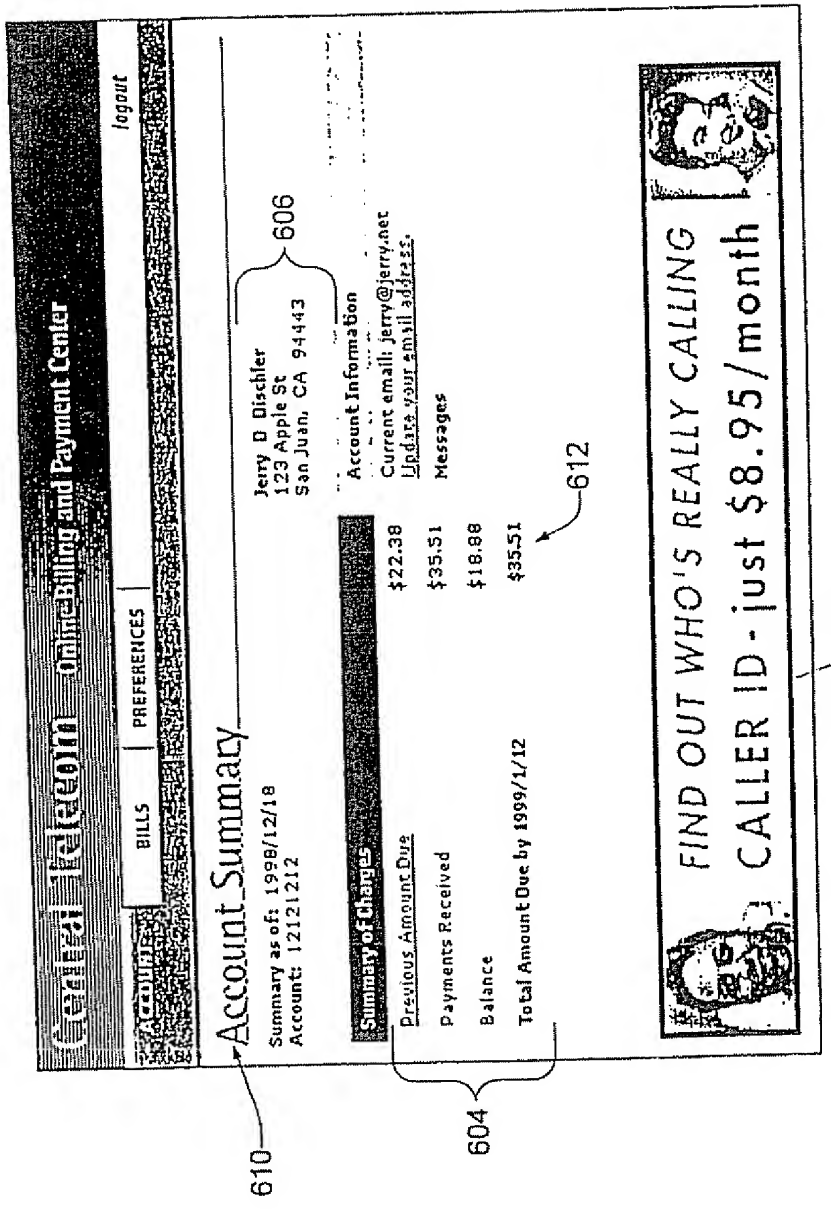

Products & Services
Business Solutions
Company Information
Contact
Introducing Online Billing
View your account and pay your bills online!
You can now, in Central Telecom Online Billing right now and get immediate access to your latest bill. Registered users can login below.
User ID
Password
LOGIN
Did you
502
504

Fig. 5

Fig. 6


Central Telecom Online Billing and Payment Center
logout
BILLS
PREFERENCES
Account Summary
Summary as of: 1998/12/18
Account: 12121212
Jerry D Dischler
123 Apple St
San Juan, CA 94443
606
Account Information
Current email: jerry@jerry.net
Update your email address.
Messages
Summary of Charges
Previous Amount Due $22.38
Payments Received $35.51
Balance $18.88
Total Amount Due by 1999/1/12 $35.51
612
610
604
FIND OUT WHO'S REALLY CALLING
CALLER ID - just $8.95/month
602

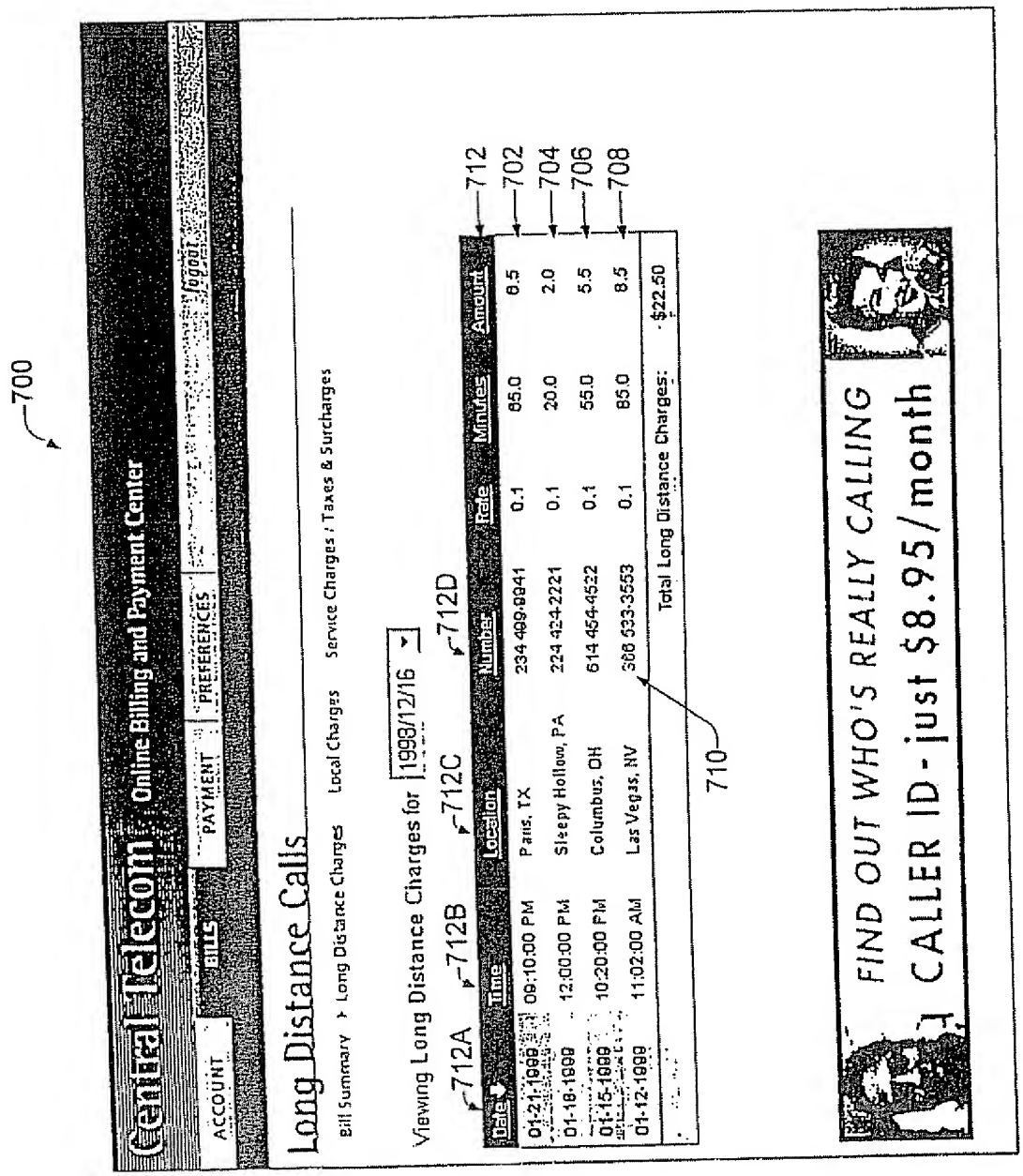

700
Central Telecom
Online Billing and Payment Center
ACCOUNT
PAYMENT
PREFERENCES
Long Distance Calls
Bill Summary ▸ Long Distance Charges
Local Charges
Service Charges / Taxes & Surcharges
Viewing Long Distance Charges for 1998/12/16
712A
712B
712C
712D
712
Date
Time
Location
Number
Rate
Minutes
Amount
01-21-1999 08:10:00 PM Paris, TX 234 409-8941 0.1 85.0 8.5
01-18-1999 12:00:00 PM Sleepy Hollow, PA 224 424-2221 0.1 20.0 2.0
01-15-1999 10:20:00 PM Columbus, OH 614 464-4522 0.1 55.0 5.5
01-12-1999 11:02:00 AM Las Vegas, NV 388 633-3553 0.1 85.0 8.5
Total Long Distance Charges: $22.50
710
702
704
706
708
FIND OUT WHO'S REALLY CALLING
CALLER ID - just $8.95/month

Fig. 7

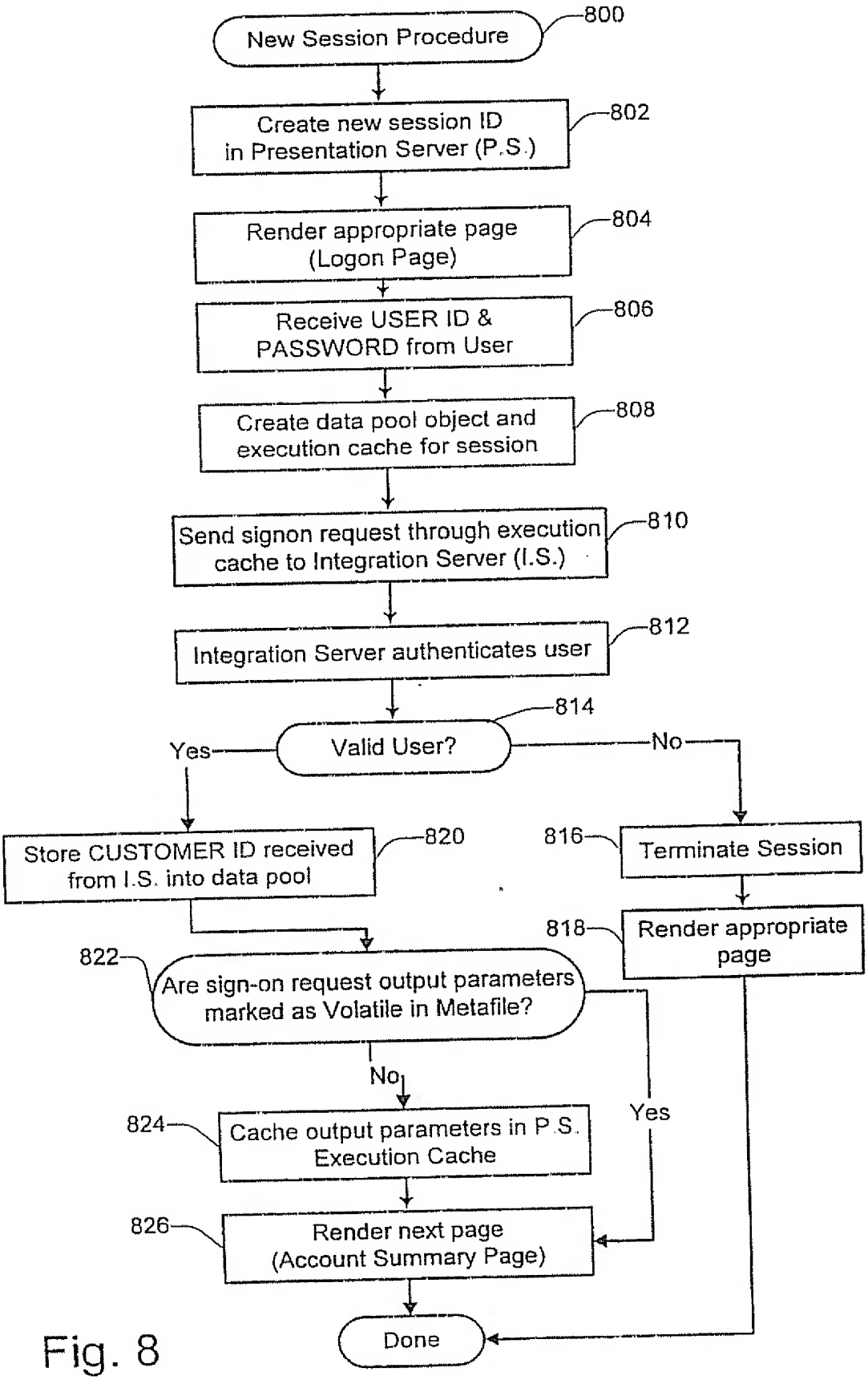

New Session Procedure
800
Create new session ID in Presentation Server (P.S.)
802
Render appropriate page (Logon Page)
804
Receive USER ID & PASSWORD from User
806
Create data pool object and execution cache for session
808
Send signon request through execution cache to Integration Server (I.S.)
810
Integration Server authenticates user
812
Valid User?
814
Yes
No
Store CUSTOMER ID received from I.S. into data pool
820
816
Terminate Session
818
Render appropriate page
822
Are sign-on request output parameters marked as Volatile in Metafile?
No
Yes
824
Cache output parameters in P.S. Execution Cache
826
Render next page (Account Summary Page)
Done

Fig. 8

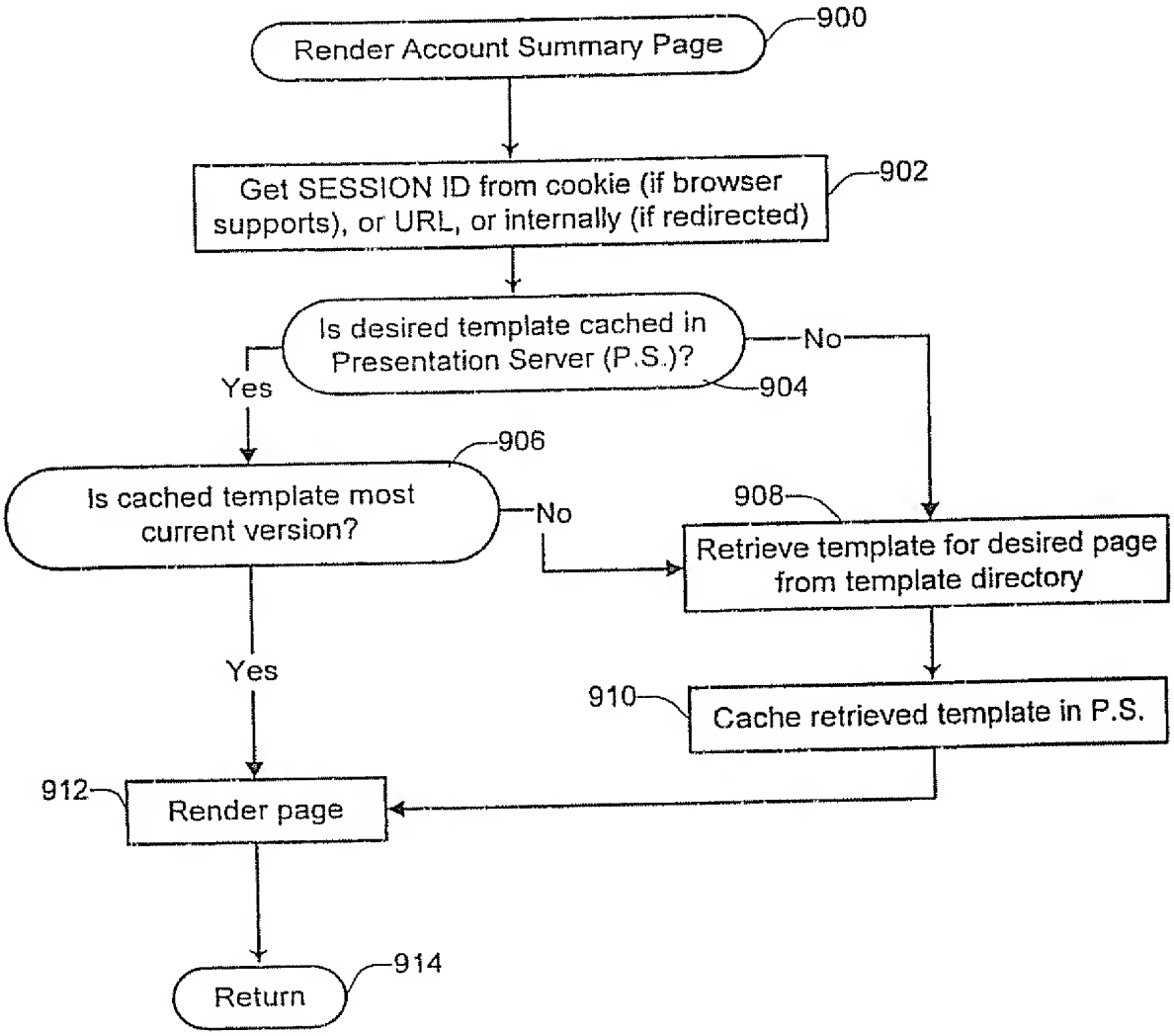
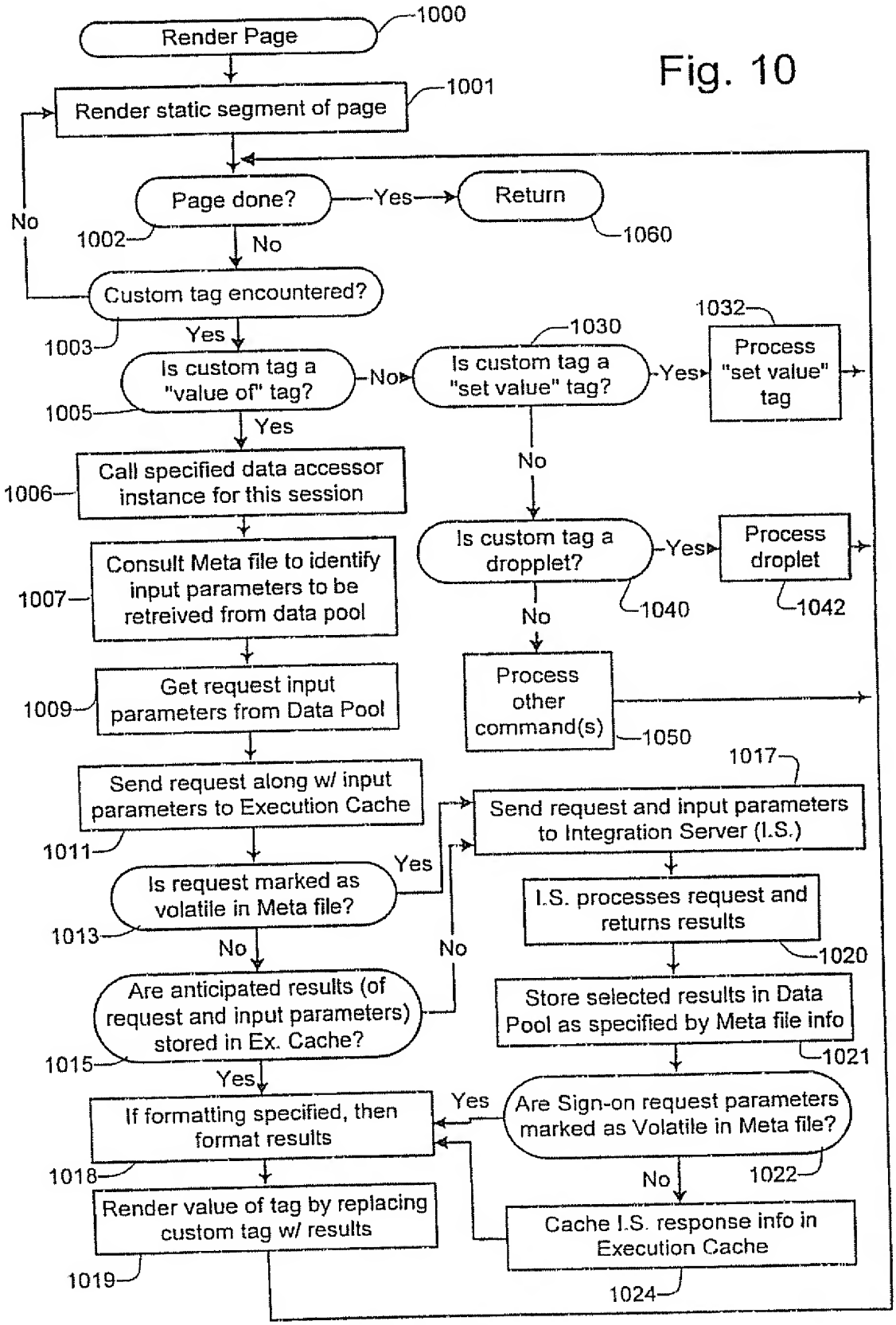

Render Account Summary Page
900
Get SESSION ID from cookie (if browser supports), or URL, or internally (if redirected)
902
Is desired template cached in Presentation Server (P.S.)?
904
No
Yes
Is cached template most current version?
906
No
Yes
908
Retrieve template for desired page from template directory
910
Cache retrieved template in P.S.
912
Render page
Return
914

Fig. 9

Fig. 10


Render Page
1000
Render static segment of page
1001
Page done?
Yes
Return
1060
No
1002
No
Custom tag encountered?
Yes
1003
Is custom tag a "value of" tag?
No
1005
Yes
Call specified data accessor instance for this session
1006
Consult Meta file to identify input parameters to be retreived from data pool
1007
Get request input parameters from Data Pool
1009
Send request along w/ input parameters to Execution Cache
1011
Is request marked as volatile in Meta file?
Yes
1013
No
Are anticipated results (of request and input parameters) stored in Ex. Cache?
No
1015
Yes
If formatting specified, then format results
1018
Render value of tag by replacing custom tag w/ results
1019
Is custom tag a "set value" tag?
1030
Yes
1032
Process "set value" tag
No
Is custom tag a dropplet?
Yes
1040
Process droplet
1042
No
Process other command(s)
1050
Send request and input parameters to Integration Server (I.S.)
1017
I.S. processes request and returns results
1020
Store selected results in Data Pool as specified by Meta file info
1021
Are Sign-on request parameters marked as Volatile in Meta file?
Yes
1022
No
Cache I.S. response info in Execution Cache
1024

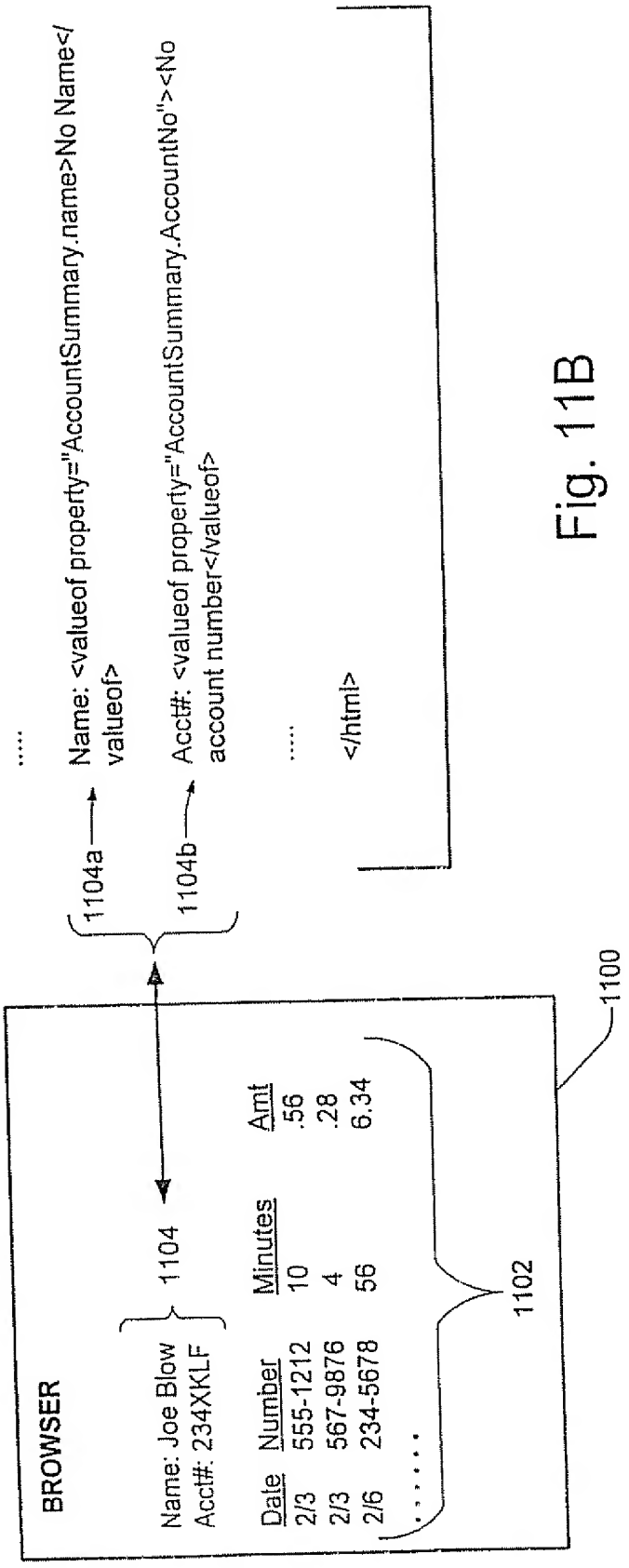
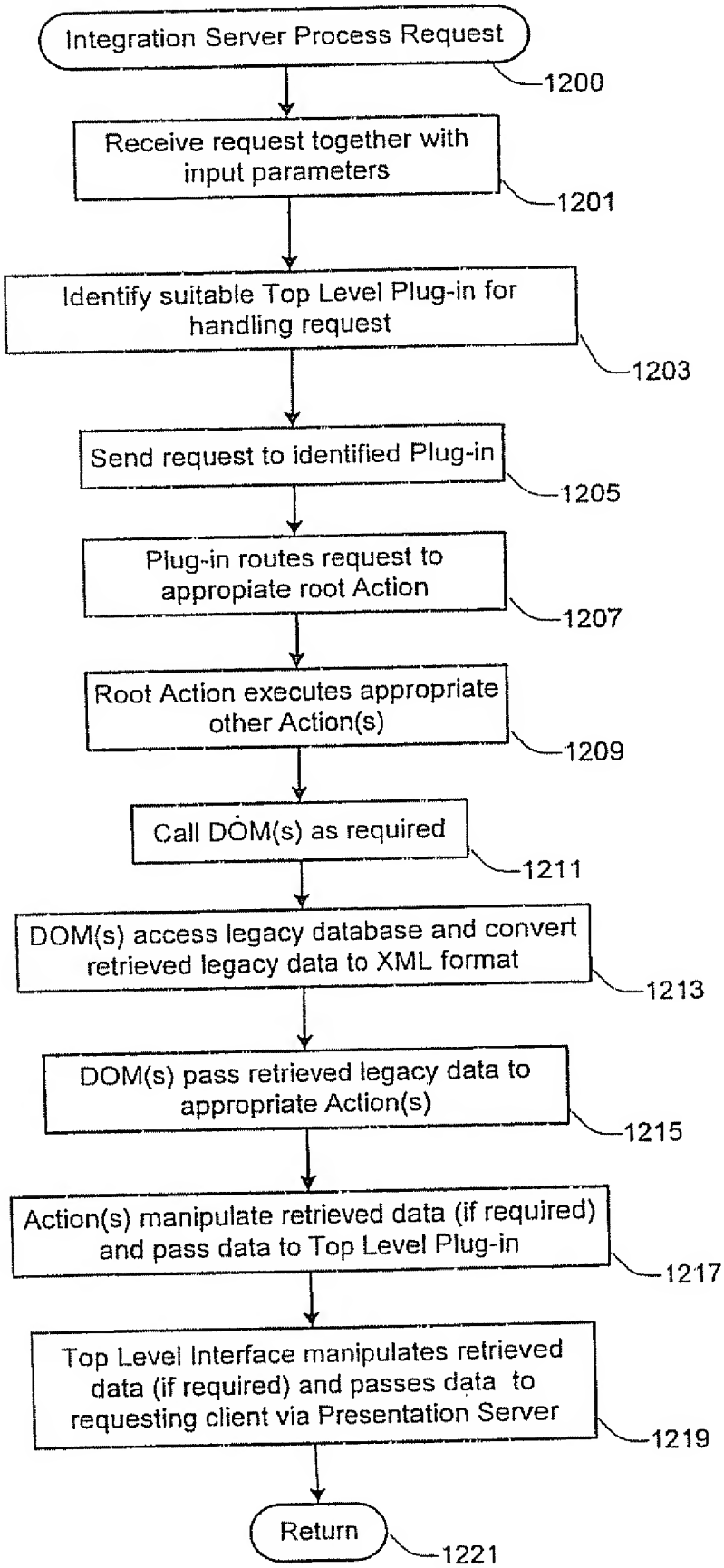


Fig. 11B

Fig. 11A

Fig. 12


Integration Server Process Request
1200
Receive request together with input parameters
1201
Identify suitable Top Level Plug-in for handling request
1203
Send request to identified Plug-in
1205
Plug-in routes request to appropiate root Action
1207
Root Action executes appropriate other Action(s)
1209
Call DOM(s) as required
1211
DOM(s) access legacy database and convert retrieved legacy data to XML format
1213
DOM(s) pass retrieved legacy data to appropriate Action(s)
1215
Action(s) manipulate retrieved data (if required) and pass data to Top Level Plug-in
1217
Top Level Interface manipulates retrieved data (if required) and passes data to requesting client via Presentation Server
1219
Return
1221

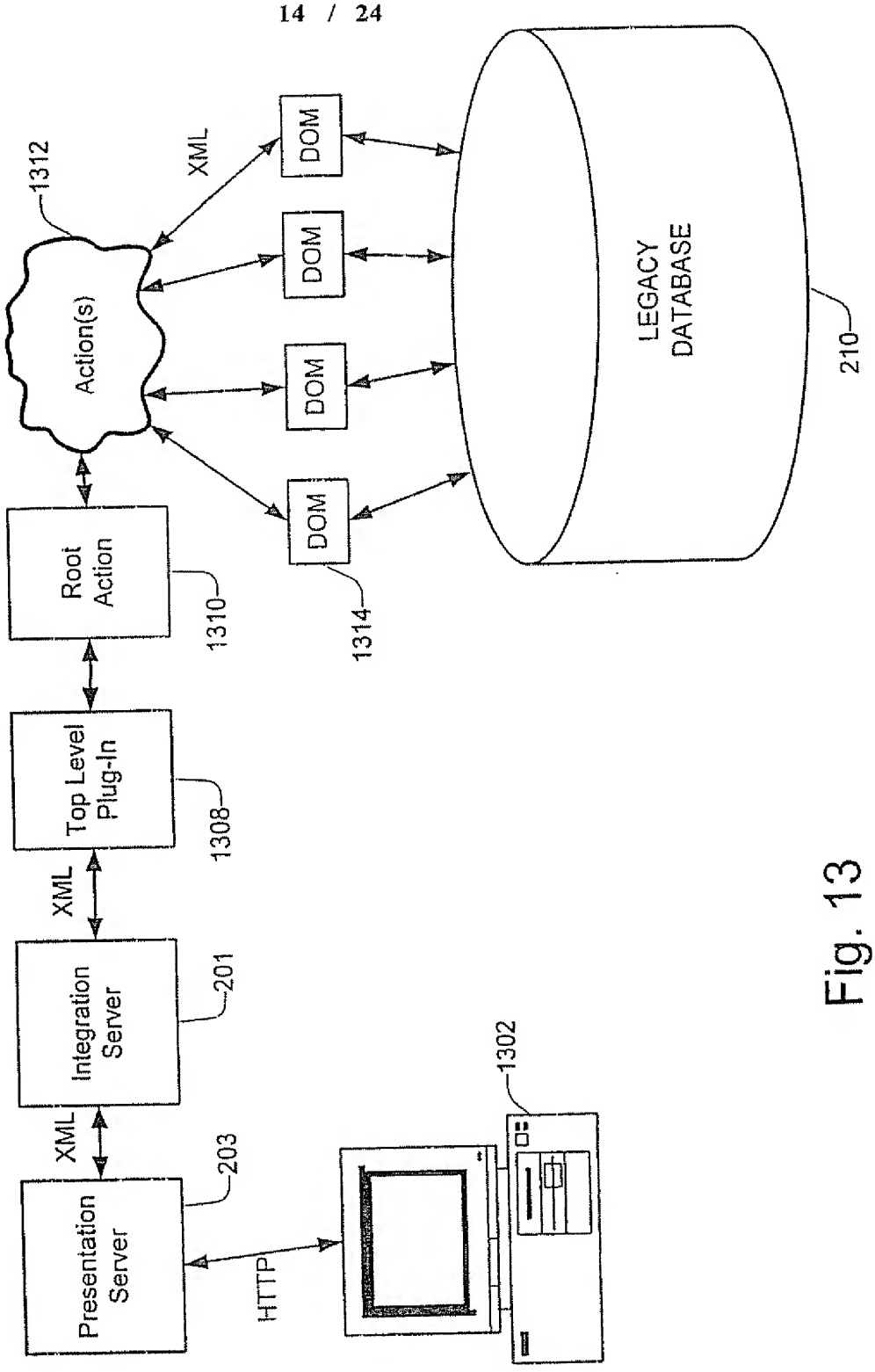

Presentation Server
203
XML
Integration Server
201
XML
Top Level Plug-In
1308
Root Action
1310
Action(s)
1312
XML
DOM
DOM
DOM
DOM
1314
LEGACY DATABASE
210
HTTP
1302

Fig. 13

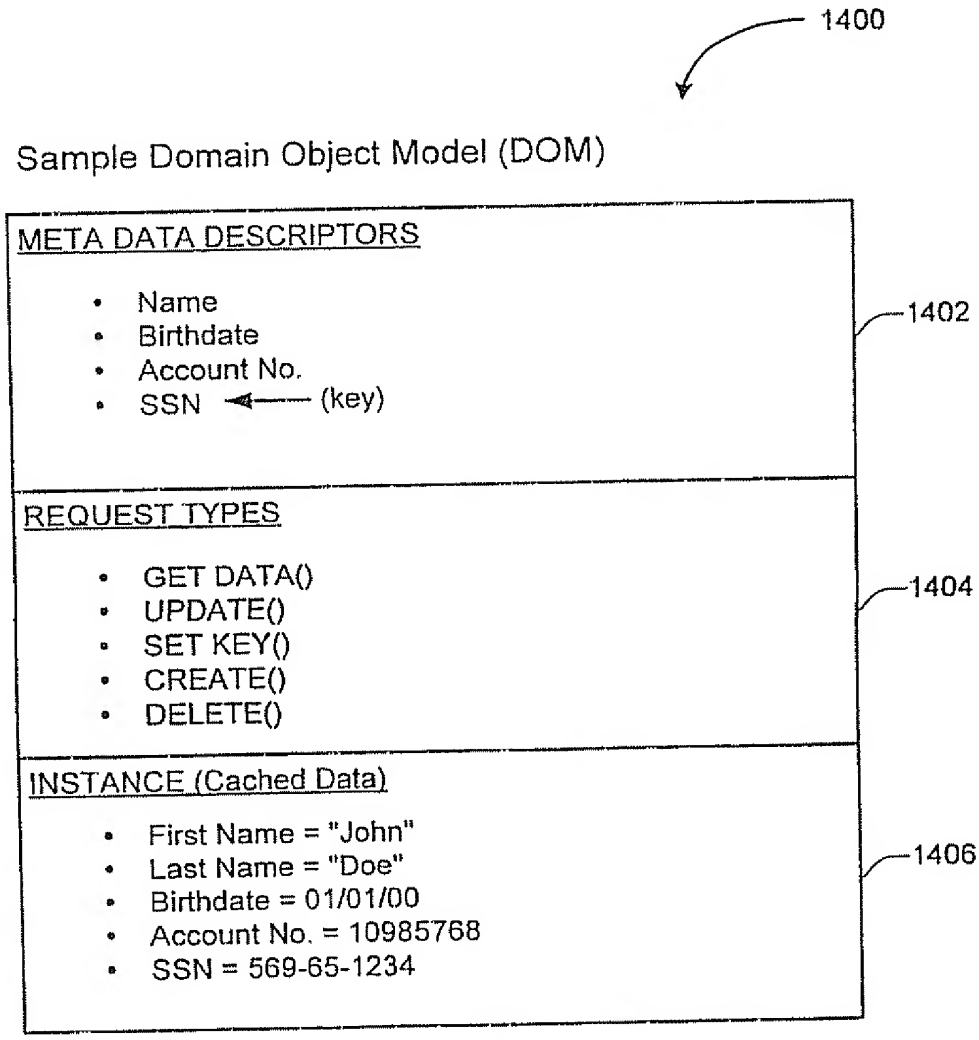

1400
Sample Domain Object Model (DOM)
META DATA DESCRIPTORS
Name
Birthdate
Account No.
SSN (key)
1402
REQUEST TYPES
GET DATA()
UPDATE()
SET KEY()
CREATE()
DELETE()
1404
INSTANCE (Cached Data)
First Name = "John"
Last Name = "Doe"
Birthdate = 01/01/00
Account No. = 10985768
SSN = 569-65-1234
1406

Fig. 14

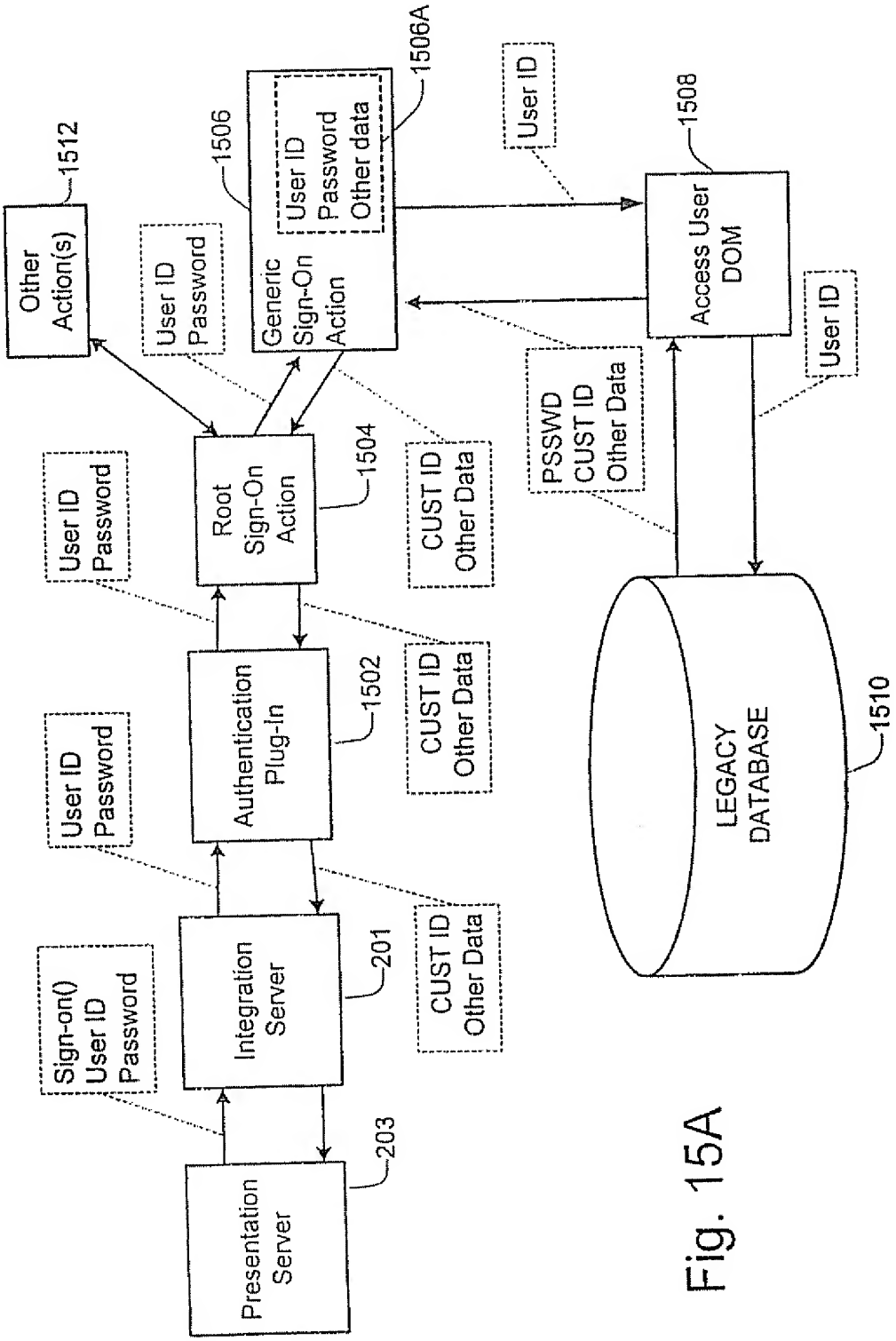

Other Action(s)
1512
User ID Password
Generic Sign-On Action
1506
User ID Password Other data
1506A
User ID
Access User DOM
1508
User ID Password
Root Sign-On Action
1504
CUST ID Other Data
PSSWD CUST ID Other Data
User ID
User ID Password
Authentication Plug-In
1502
CUST ID Other Data
LEGACY DATABASE
1510
User ID Password
Integration Server
201
CUST ID Other Data
Sign-on() User ID Password
Presentation Server
203

Fig. 15A

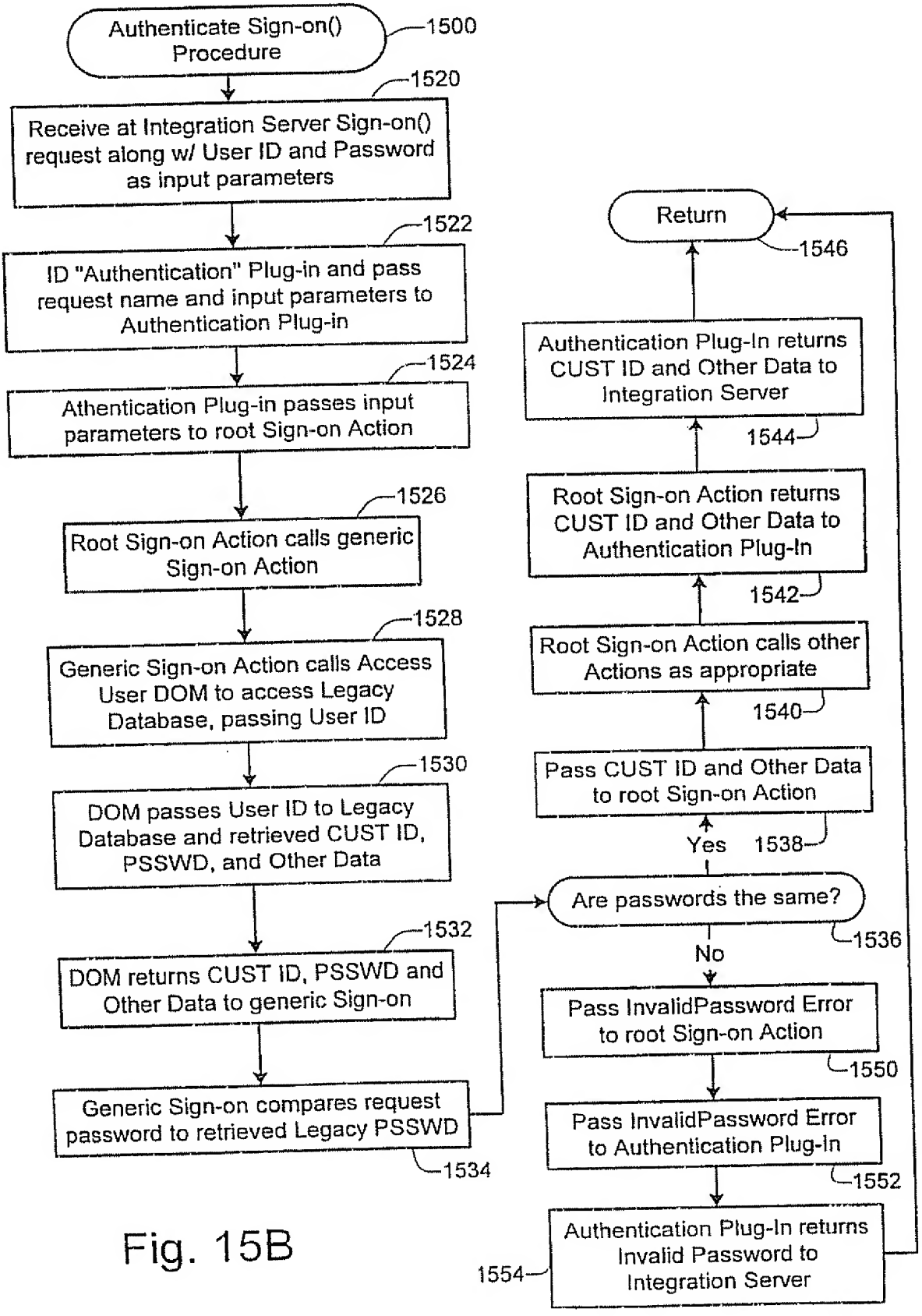

Authenticate Sign-on() Procedure
1500
1520
Receive at Integration Server Sign-on() request along w/ User ID and Password as input parameters
1522
ID "Authentication" Plug-in and pass request name and input parameters to Authentication Plug-in
1524
Athentication Plug-in passes input parameters to root Sign-on Action
1526
Root Sign-on Action calls generic Sign-on Action
1528
Generic Sign-on Action calls Access User DOM to access Legacy Database, passing User ID
1530
DOM passes User ID to Legacy Database and retrieved CUST ID, PSSWD, and Other Data
1532
DOM returns CUST ID, PSSWD and Other Data to generic Sign-on
Generic Sign-on compares request password to retrieved Legacy PSSWD
1534
Are passwords the same?
1536
No
Yes
1538
Pass CUST ID and Other Data to root Sign-on Action
1540
Root Sign-on Action calls other Actions as appropriate
1542
Root Sign-on Action returns CUST ID and Other Data to Authentication Plug-In
1544
Authentication Plug-In returns CUST ID and Other Data to Integration Server
Return
1546
Pass InvalidPassword Error to root Sign-on Action
1550
Pass InvalidPassword Error to Authentication Plug-In
1552
Authentication Plug-In returns Invalid Password to Integration Server
1554

Fig. 15B

SAMPLE META INFORMATION

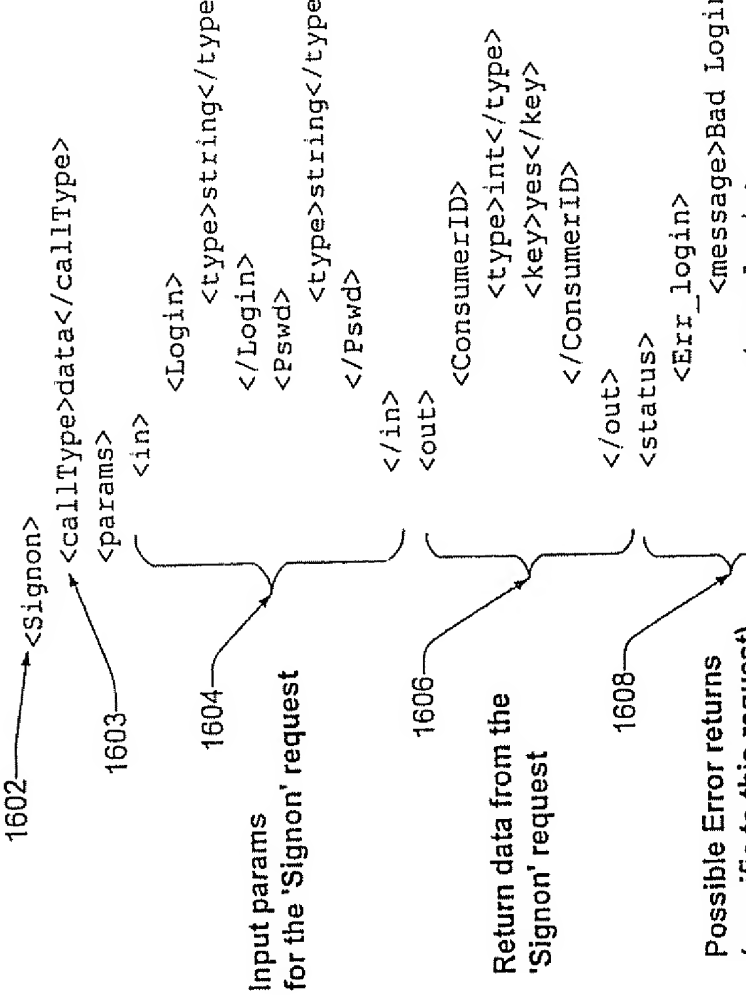


Fig. 16

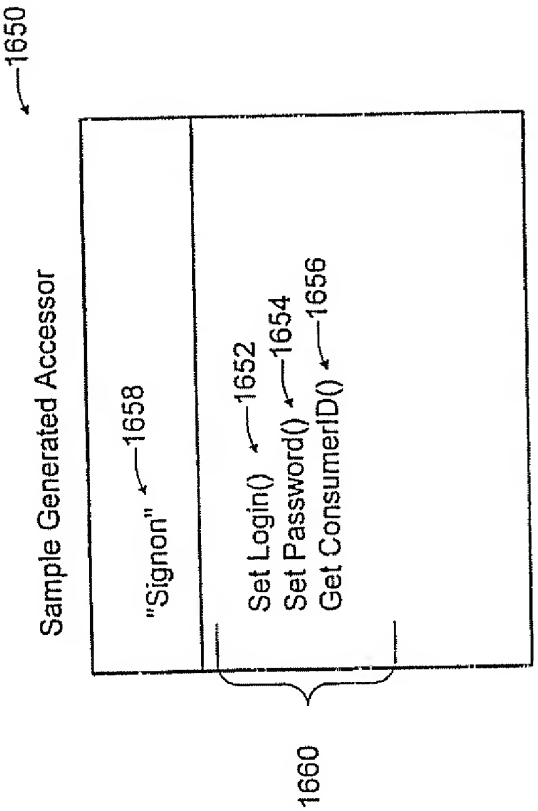
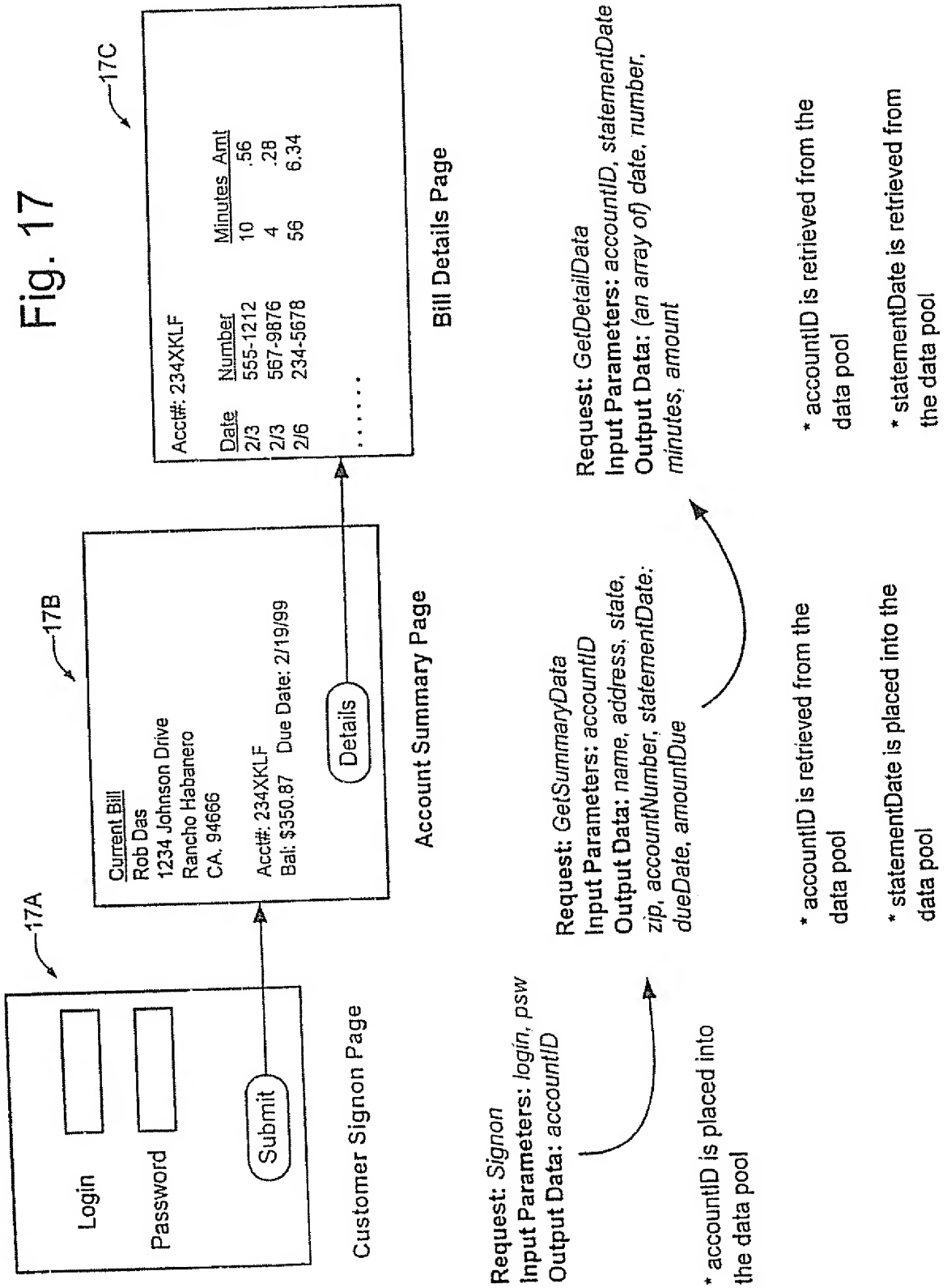

1650
Sample Generated Accessor
"Signon"
1658
Set Login()
Set Password()
Get ConsumerID()
1652
1654
1656
1660

Fig. 16A

Fig. 17


17A
Login
Password
Submit
Customer Signon Page
17B
Current Bill
Rob Das
1234 Johnson Drive
Rancho Habanero
CA. 94666
Acct#: 234XKLF
Bal: $350.87 Due Date: 2/19/99
Details
Account Summary Page
17C
Acct#: 234XKLF
Date Number Minutes Amt
2/3 555-1212 10 .56
2/3 567-9876 4 .28
2/6 234-5678 56 6.34
.....
Bill Details Page
Request: Signon
Input Parameters: login, psw
Output Data: accountID
* accountID is placed into the data pool
Request: GetSummaryData
Input Parameters: accountID
Output Data: name, address, state, zip, accountNumber, statementDate, dueDate, amountDue
* accountID is retrieved from the data pool
* statementDate is placed into the data pool
Request: GetDetailData
Input Parameters: accountID, statementDate
Output Data: (an array of) date, number, minutes, amount
* accountID is retrieved from the data pool
* statementDate is retrieved from the data pool

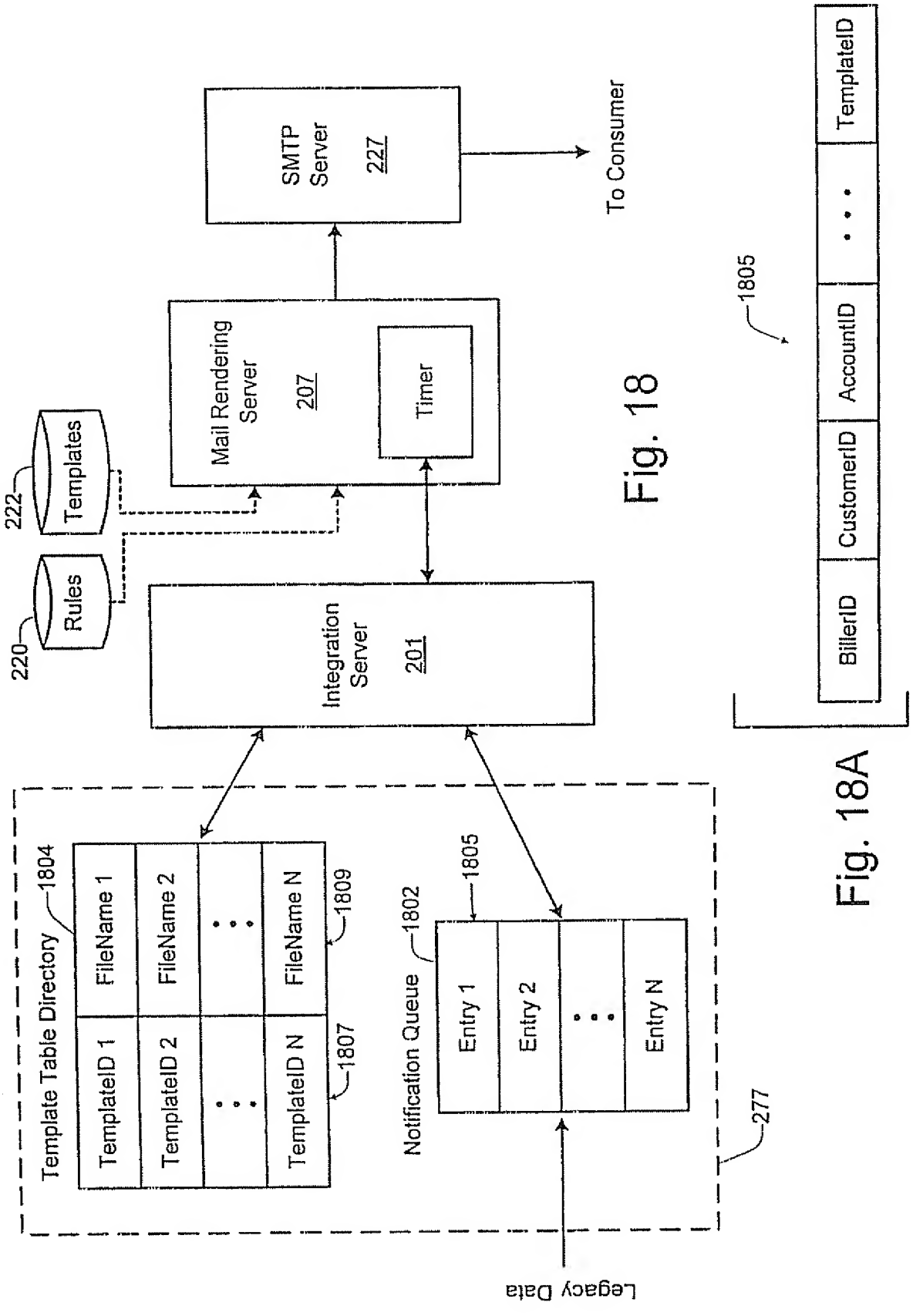


Fig. 18

Fig. 18A

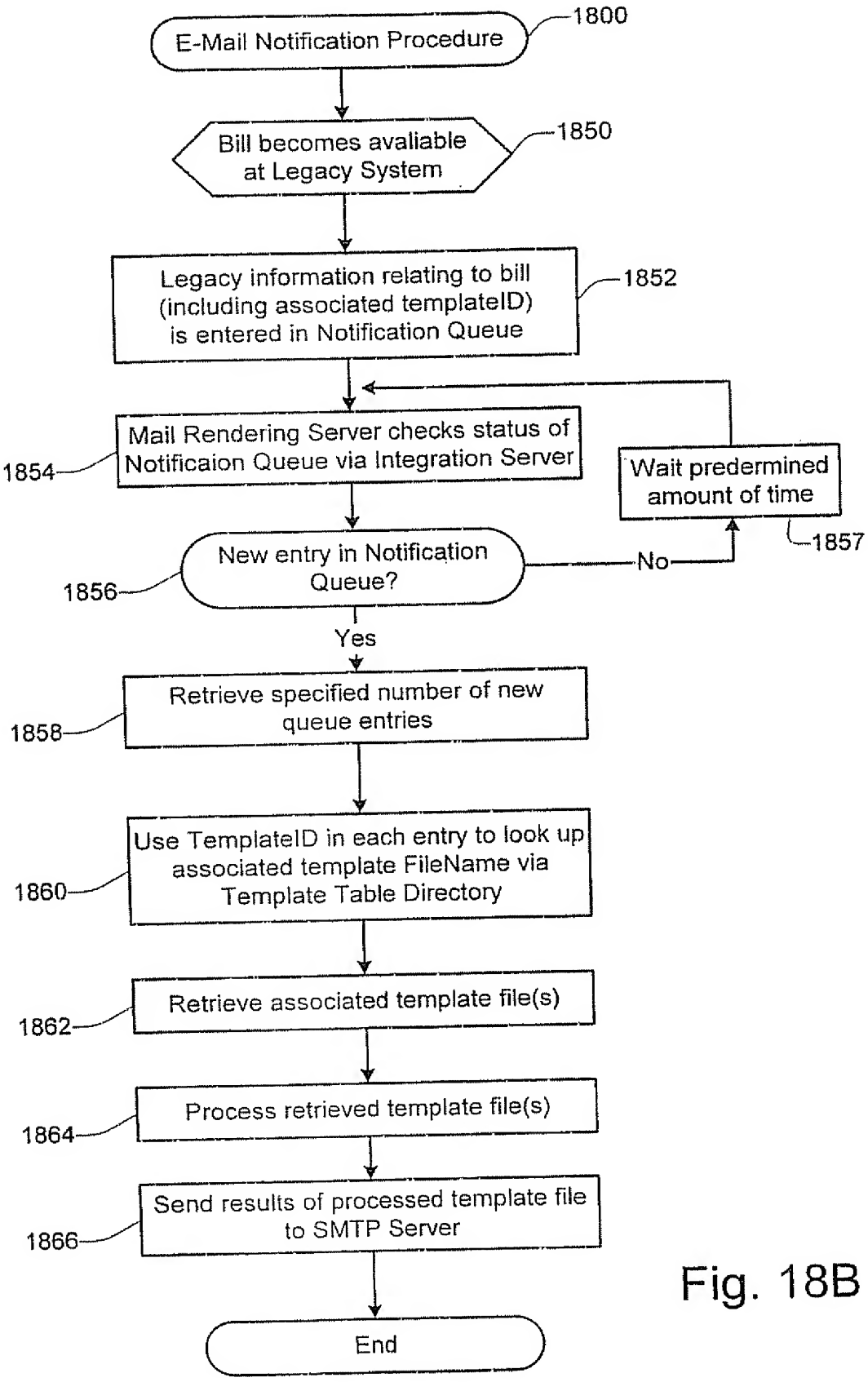
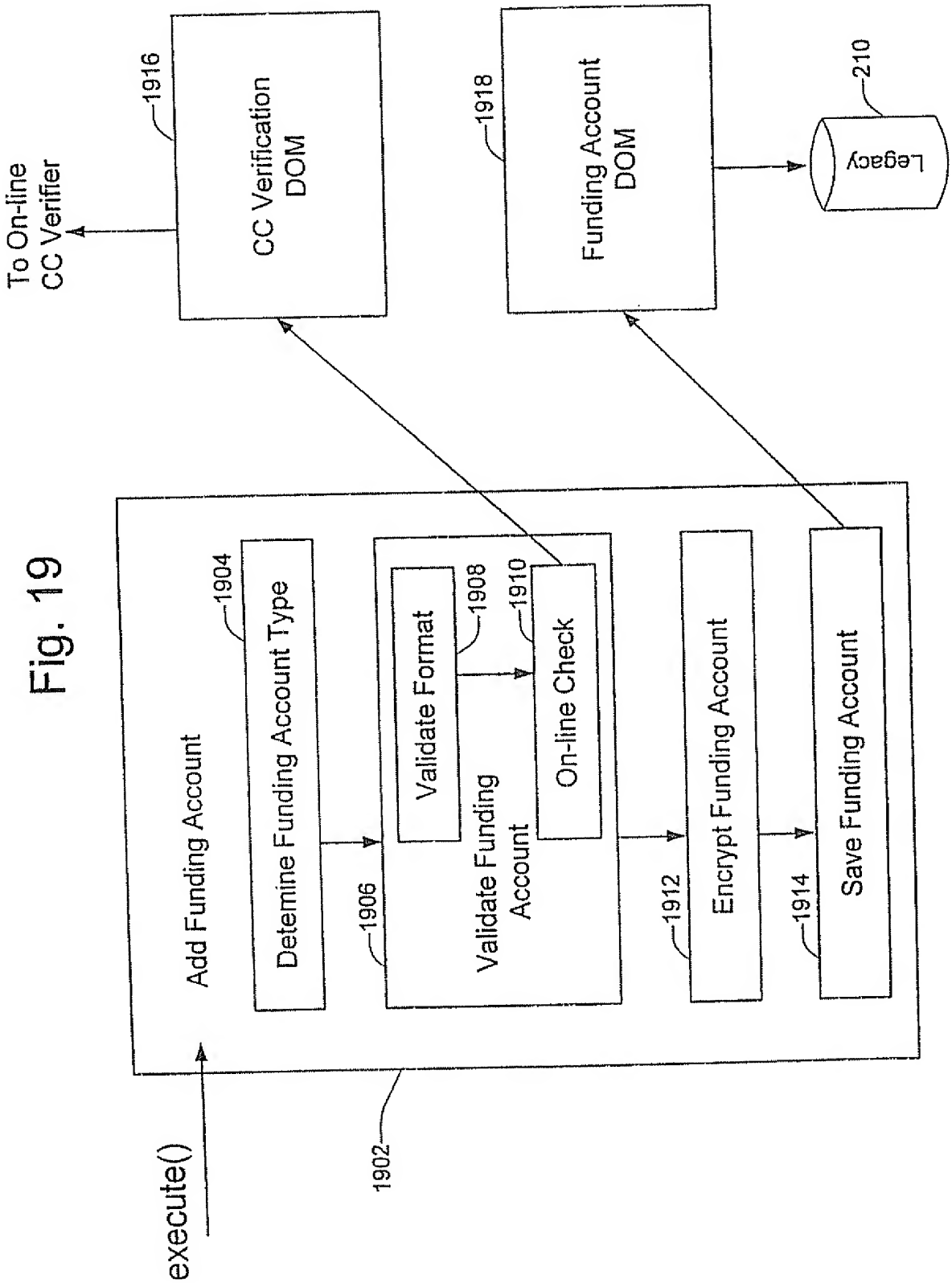

E-Mail Notification Procedure
1800
Bill becomes avaliable at Legacy System
1850
Legacy information relating to bill (including associated templateID) is entered in Notification Queue
1852
Mail Rendering Server checks status of Notificaion Queue via Integration Server
1854
Wait predermined amount of time
1857
New entry in Notification Queue?
1856
No
Yes
Retrieve specified number of new queue entries
1858
Use TemplateID in each entry to look up associated template FileName via Template Table Directory
1860
Retrieve associated template file(s)
1862
Process retrieved template file(s)
1864
Send results of processed template file to SMTP Server
1866
End

Fig. 18B

Fig. 19


execute()
1902
Add Funding Account
1904
Detemine Funding Account Type
1906
Validate Funding Account
Validate Format
1908
1910
On-line Check
1912
Encrypt Funding Account
1914
Save Funding Account
To On-line CC Verifier
1916
CC Verification DOM
1918
Funding Account DOM
210
Legacy

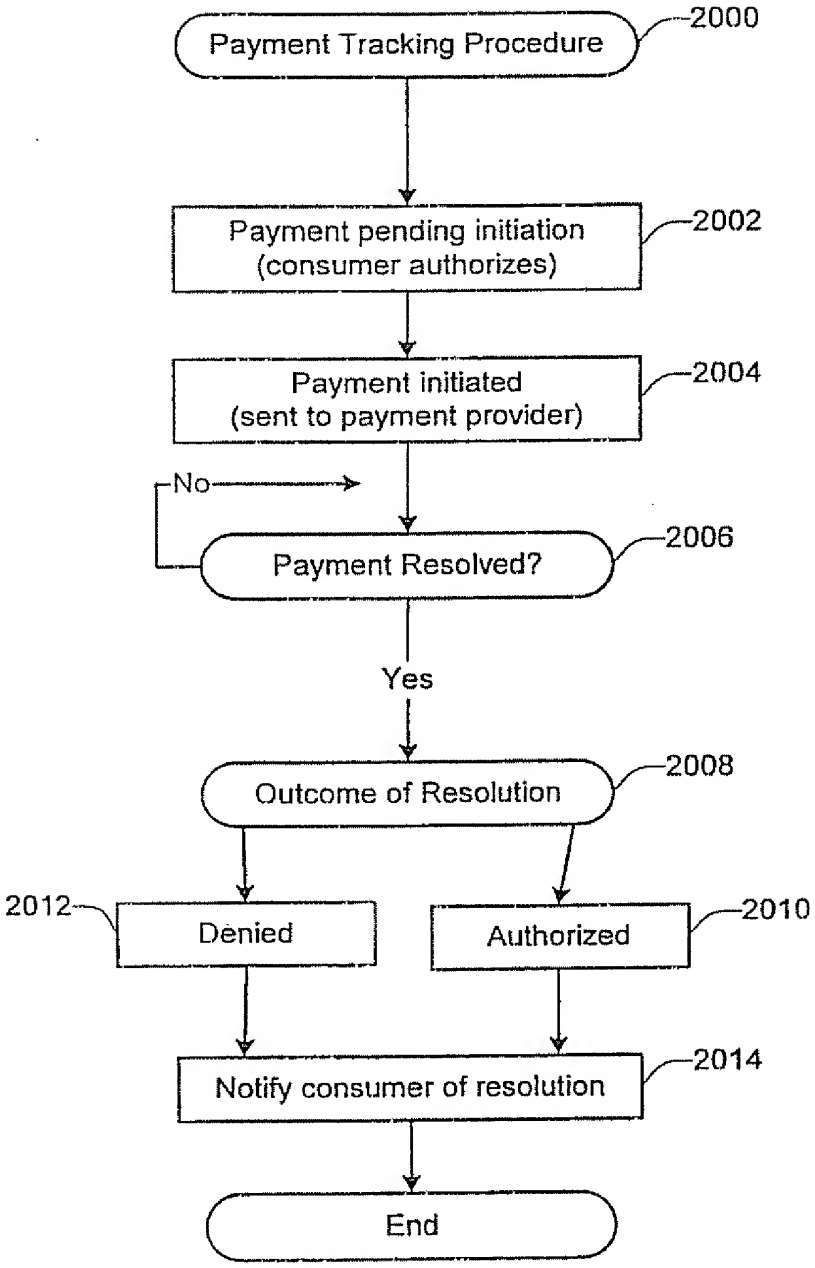

Payment Tracking Procedure
2000
Payment pending initiation
(consumer authorizes)
2002
Payment initiated
(sent to payment provider)
2004
No
Payment Resolved?
2006
Yes
Outcome of Resolution
2008
2012
Denied
Authorized
2010
Notify consumer of resolution
2014
End

Fig. 20